# React.js Notes – Part 2

Contents

[React.js Notes – Part 2 1](#_Toc115166838)

[Section 5: Rendering Lists & Conditional Content 1](#_Toc115166839)

[63. Module Introduction 2](#_Toc115166840)

[64. Rendering Lists of Data 2](#_Toc115166841)

[65. Using Stateful Lists 29](#_Toc115166842)

[66. Understanding "Keys" 35](#_Toc115166843)

[Assignment 3: Time to Practice: Working with Lists 42](#_Toc115166844)

[67. Outputting Conditional Content 42](#_Toc115166845)

[68. Adding Conditional Return Statements 47](#_Toc115166846)

[Assignment 4: Time to Practice: Conditional Content 51](#_Toc115166847)

[69. Demo App: Adding a Chart 64](#_Toc115166848)

[70. Adding Dynamic Styles 69](#_Toc115166849)

[71. Wrap Up & Next Steps 80](#_Toc115166850)

[72. Fixing a Small Bug 89](#_Toc115166851)

[Section 6: Styling React Components 89](#_Toc115166852)

[74. Module Introduction 89](#_Toc115166853)

[75. Setting Dynamic Inline Styles 92](#_Toc115166854)

[76. Setting CSS Classes Dynamically 105](#_Toc115166855)

[77. Introducing Styled Components 115](#_Toc115166856)

[78. Styled Components & Dynamic Props 125](#_Toc115166857)

[79. Styled Components & Media Queries 141](#_Toc115166858)

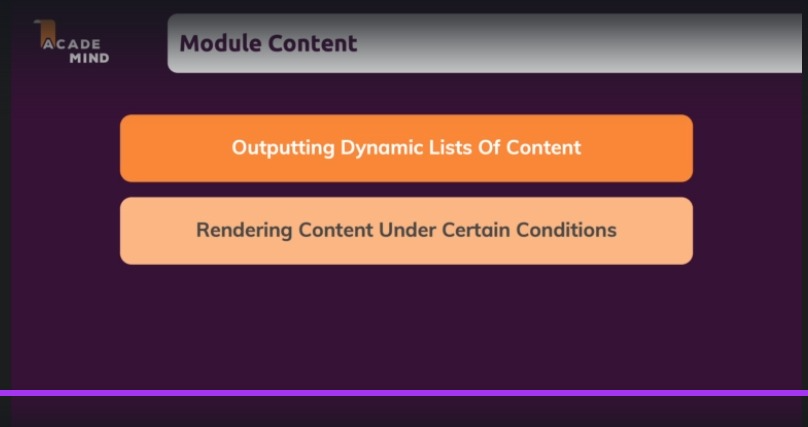
## Section 5: Rendering Lists & Conditional Content

### 63. Module Introduction

In this section, we will dive into rendering lists and conditional content as well as working with really dynamic content. We will look at how we can output arrays of data on our page, and how we can show different content based on different conditions.



This is what we will look at in this module. We will continue working on the application we already started over the last course sections, but in this module, we will finish it by adding these features.



### 64. Rendering Lists of Data

Our list is static currently, not dynamic, meaning that as we add a new item to the form, it's not added to the list.

We want to render our expenses which are defined in App.js. We have an expenses array. We want to render this in the Expenses component. Therefore, the first step is to pass our expenses down via props, which we are already doing. We are passing the items prop,

|  |
| --- |
| src/App.js |
| import React from "react";  import NewExpense from "./components/NewExpense/NewExpense";  import Expenses from "./components/Expenses/Expenses";  const App = () => {  const expenses = [  {  id: "e1",  title: "Toilet Paper",  amount: 94.12,  date: new Date(2020, 7, 14),  },  { id: "e2", title: "New TV", amount: 799.49, date: new Date(2021, 2, 12) },  {  id: "e3",  title: "Car Insurance",  amount: 294.67,  date: new Date(2021, 2, 28),  },  {  id: "e4",  title: "New Desk (Wooden)",  amount: 450,  date: new Date(2021, 5, 12),  },  ];  const addExpenseHandler = expense => {  console.log('In App.js');  console.log(expense);  }  return (  <div>  <NewExpense onAddExpense={addExpenseHandler}/>  <Expenses **items**={expenses} />  </div>  );  };  export default App; |

which points at this expenses array.

|  |
| --- |
| src/App.js |
| import React from "react";  import NewExpense from "./components/NewExpense/NewExpense";  import Expenses from "./components/Expenses/Expenses";  const App = () => {  const expenses = [  {  id: "e1",  title: "Toilet Paper",  amount: 94.12,  date: new Date(2020, 7, 14),  },  { id: "e2", title: "New TV", amount: 799.49, date: new Date(2021, 2, 12) },  {  id: "e3",  title: "Car Insurance",  amount: 294.67,  date: new Date(2021, 2, 28),  },  {  id: "e4",  title: "New Desk (Wooden)",  amount: 450,  date: new Date(2021, 5, 12),  },  ];  const addExpenseHandler = expense => {  console.log('In App.js');  console.log(expense);  }  return (  <div>  <NewExpense onAddExpense={addExpenseHandler}/>  <Expenses items={**expenses**} />  </div>  );  };  export default App; |

So, in the Expenses component, we do get our list of expenses, but we are currently not using that, and that is what we want to change.

|  |
| --- |
| src/components/Expenses/Expenses.js |
| import React, {useState} from "react";  import ExpenseItem from "./ExpenseItem";  import ExpensesFilter from "./ExpensesFilter";  import Card from "../UI/Card";  import "./Expenses.css";  const Expenses = (props) => {  const [filteredYear, setFilteredYear] = useState('2020');  const filterChangeHandler = selectedYear => {  setFilteredYear(selectedYear);  }  return (  <div>  <Card className="expenses">  <ExpensesFilter selected={filteredYear} onChangeFilter={filterChangeHandler}/>  <ExpenseItem  title={props.items[0].title}  amount={props.items[0].amount}  date={props.items[0].date}  />  <ExpenseItem  title={props.items[1].title}  amount={props.items[1].amount}  date={props.items[1].date}  />  <ExpenseItem  title={props.items[2].title}  amount={props.items[2].amount}  date={props.items[2].date}  />  <ExpenseItem  title={props.items[3].title}  amount={props.items[3].amount}  date={props.items[3].date}  />  </Card>  </div>  );  };  export default Expenses; |

On our props, we have this items prop and the value of that will be that array,

|  |
| --- |
| src/App.js |
| import React from "react";  import NewExpense from "./components/NewExpense/NewExpense";  import Expenses from "./components/Expenses/Expenses";  const App = () => {  const expenses = [  {  id: "e1",  title: "Toilet Paper",  amount: 94.12,  date: new Date(2020, 7, 14),  },  { id: "e2", title: "New TV", amount: 799.49, date: new Date(2021, 2, 12) },  {  id: "e3",  title: "Car Insurance",  amount: 294.67,  date: new Date(2021, 2, 28),  },  {  id: "e4",  title: "New Desk (Wooden)",  amount: 450,  date: new Date(2021, 5, 12),  },  ];  const addExpenseHandler = expense => {  console.log('In App.js');  console.log(expense);  }  return (  <div>  <NewExpense onAddExpense={addExpenseHandler}/>  <Expenses **items**={expenses} />  </div>  );  };  export default App; |

which we in the end want to render here. But we don't want to output the array as text or anything like that.

|  |
| --- |
| src/components/Expenses/Expenses.js |
| import React, {useState} from "react";  import ExpenseItem from "./ExpenseItem";  import ExpensesFilter from "./ExpensesFilter";  import Card from "../UI/Card";  import "./Expenses.css";  const Expenses = (props) => {  const [filteredYear, setFilteredYear] = useState('2020');  const filterChangeHandler = selectedYear => {  setFilteredYear(selectedYear);  }  return (  <div>  <Card className="expenses">  <ExpensesFilter selected={filteredYear} onChangeFilter={filterChangeHandler}/>  **<ExpenseItem**  **title={props.items[0].title}**  **amount={props.items[0].amount}**  **date={props.items[0].date}**  **/>**  **<ExpenseItem**  **title={props.items[1].title}**  **amount={props.items[1].amount}**  **date={props.items[1].date}**  **/>**  **<ExpenseItem**  **title={props.items[2].title}**  **amount={props.items[2].amount}**  **date={props.items[2].date}**  **/>**  **<ExpenseItem**  **title={props.items[3].title}**  **amount={props.items[3].amount}**  **date={props.items[3].date}**  **/>**  </Card>  </div>  );  };  export default Expenses; |

Instead, we want to render one ExpenseItem per element in the array, and that's straightforward to do.

For this, we first of all need single curly braces, opening and closing, because we want to execute a dynamic expression in our JSX code, and that is done with curly braces. We can execute JavaScript expressions inside of the curly braces.

|  |
| --- |
| src/components/Expenses/Expenses.js |
| import React, {useState} from "react";  import ExpenseItem from "./ExpenseItem";  import ExpensesFilter from "./ExpensesFilter";  import Card from "../UI/Card";  import "./Expenses.css";  const Expenses = (props) => {  const [filteredYear, setFilteredYear] = useState('2020');  const filterChangeHandler = selectedYear => {  setFilteredYear(selectedYear);  }  return (  <div>  <Card className="expenses">  <ExpensesFilter selected={filteredYear} onChangeFilter={filterChangeHandler}/>  **{}**  <ExpenseItem  title={props.items[0].title}  amount={props.items[0].amount}  date={props.items[0].date}  />  <ExpenseItem  title={props.items[1].title}  amount={props.items[1].amount}  date={props.items[1].date}  />  <ExpenseItem  title={props.items[2].title}  amount={props.items[2].amount}  date={props.items[2].date}  />  <ExpenseItem  title={props.items[3].title}  amount={props.items[3].amount}  date={props.items[3].date}  />  </Card>  </div>  );  };  export default Expenses; |

The expression which I do want to execute here is that when I reach out to my array of data, in this case the array of expenses, for every element in the array, we want to create such a JSX element.

|  |
| --- |
| src/components/Expenses/Expenses.js |
| import React, {useState} from "react";  import ExpenseItem from "./ExpenseItem";  import ExpensesFilter from "./ExpensesFilter";  import Card from "../UI/Card";  import "./Expenses.css";  const Expenses = (props) => {  const [filteredYear, setFilteredYear] = useState('2020');  const filterChangeHandler = selectedYear => {  setFilteredYear(selectedYear);  }  return (  <div>  <Card className="expenses">  <ExpensesFilter selected={filteredYear} onChangeFilter={filterChangeHandler}/>  {}  **<ExpenseItem**  **title={props.items[0].title}**  **amount={props.items[0].amount}**  **date={props.items[0].date}**  **/>**  <ExpenseItem  title={props.items[1].title}  amount={props.items[1].amount}  date={props.items[1].date}  />  <ExpenseItem  title={props.items[2].title}  amount={props.items[2].amount}  date={props.items[2].date}  />  <ExpenseItem  title={props.items[3].title}  amount={props.items[3].amount}  date={props.items[3].date}  />  </Card>  </div>  );  };  export default Expenses; |

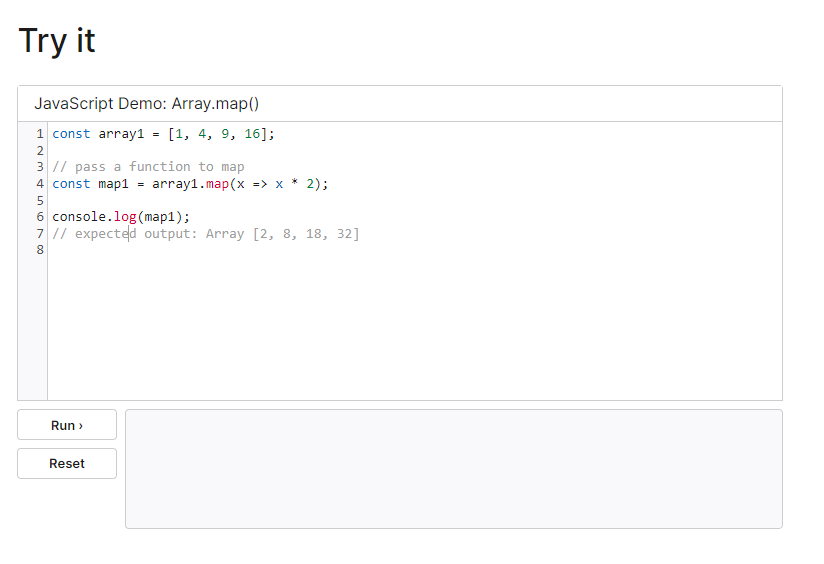
For this, we can access props.items (inside of the curly braces),

|  |
| --- |
| src/components/Expenses/Expenses.js |
| import React, {useState} from "react";  import ExpenseItem from "./ExpenseItem";  import ExpensesFilter from "./ExpensesFilter";  import Card from "../UI/Card";  import "./Expenses.css";  const Expenses = (props) => {  const [filteredYear, setFilteredYear] = useState('2020');  const filterChangeHandler = selectedYear => {  setFilteredYear(selectedYear);  }  return (  <div>  <Card className="expenses">  <ExpensesFilter selected={filteredYear} onChangeFilter={filterChangeHandler}/>  {**props.items**}  <ExpenseItem  title={props.items[0].title}  amount={props.items[0].amount}  date={props.items[0].date}  />  <ExpenseItem  title={props.items[1].title}  amount={props.items[1].amount}  date={props.items[1].date}  />  <ExpenseItem  title={props.items[2].title}  amount={props.items[2].amount}  date={props.items[2].date}  />  <ExpenseItem  title={props.items[3].title}  amount={props.items[3].amount}  date={props.items[3].date}  />  </Card>  </div>  );  };  export default Expenses; |

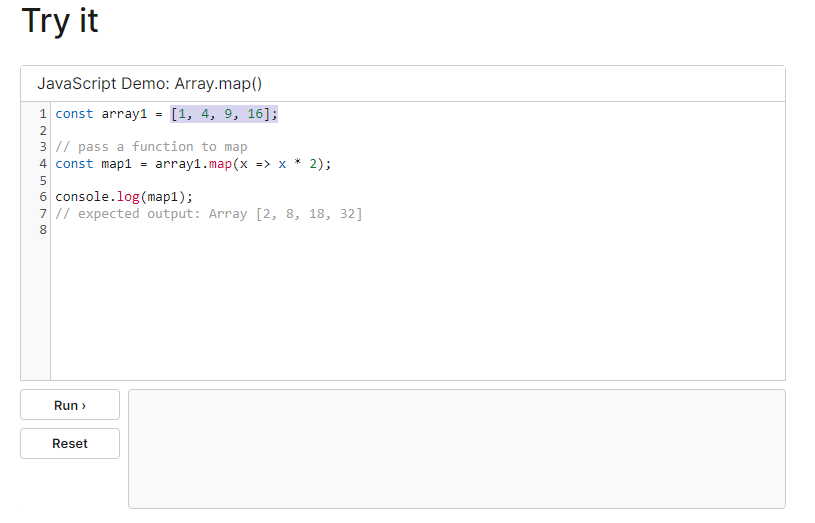
That's our array of items, our array of expenses, and now can use a built in array method, which is built into standard JavaScript, and that's the map method.

|  |
| --- |
| src/components/Expenses/Expenses.js |
| import React, {useState} from "react";  import ExpenseItem from "./ExpenseItem";  import ExpensesFilter from "./ExpensesFilter";  import Card from "../UI/Card";  import "./Expenses.css";  const Expenses = (props) => {  const [filteredYear, setFilteredYear] = useState('2020');  const filterChangeHandler = selectedYear => {  setFilteredYear(selectedYear);  }  return (  <div>  <Card className="expenses">  <ExpensesFilter selected={filteredYear} onChangeFilter={filterChangeHandler}/>  {props.items**.map()**}  <ExpenseItem  title={props.items[0].title}  amount={props.items[0].amount}  date={props.items[0].date}  />  <ExpenseItem  title={props.items[1].title}  amount={props.items[1].amount}  date={props.items[1].date}  />  <ExpenseItem  title={props.items[2].title}  amount={props.items[2].amount}  date={props.items[2].date}  />  <ExpenseItem  title={props.items[3].title}  amount={props.items[3].amount}  date={props.items[3].date}  />  </Card>  </div>  );  };  export default Expenses; |

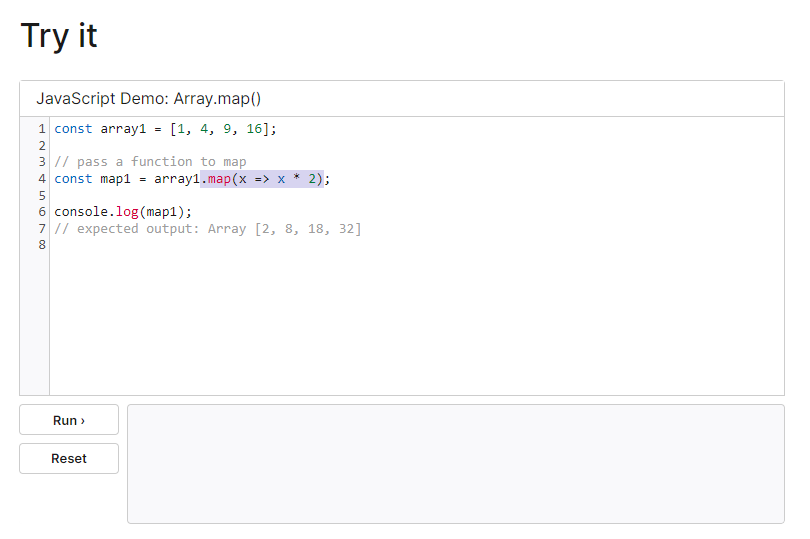
The map() method creates a new array based on another array, and that basically transforms every element in that original array. If we look at some [documentation for the map() method](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/map), we see the following example:



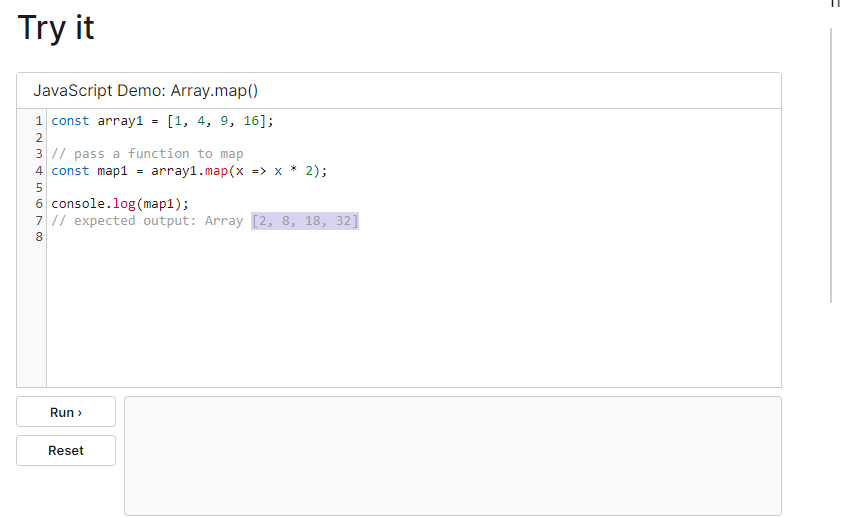
We have an array of numbers,



and with map(), we multiply every number with two,



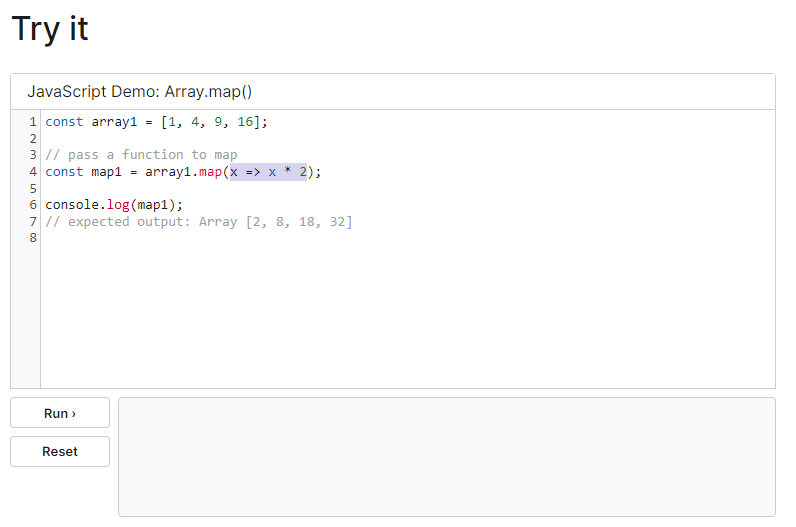
so that we get this new array as a result.



For applying this transformation, map(), this method which we can call,

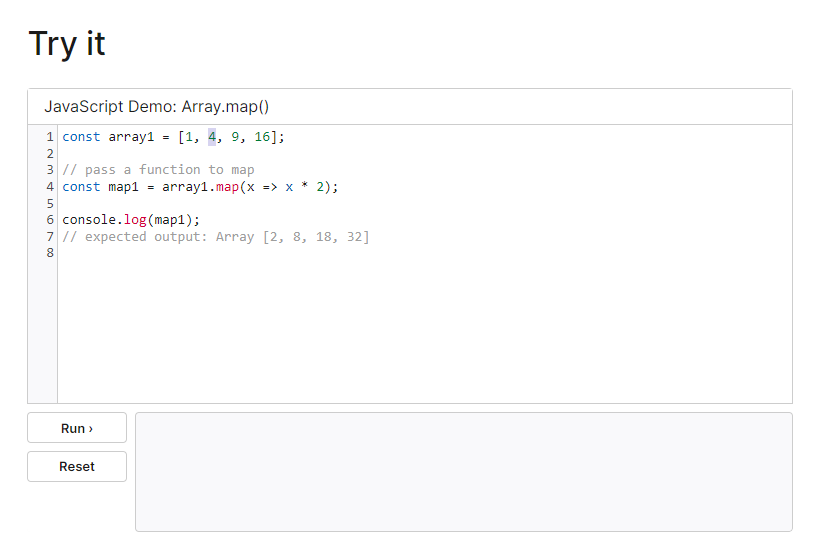


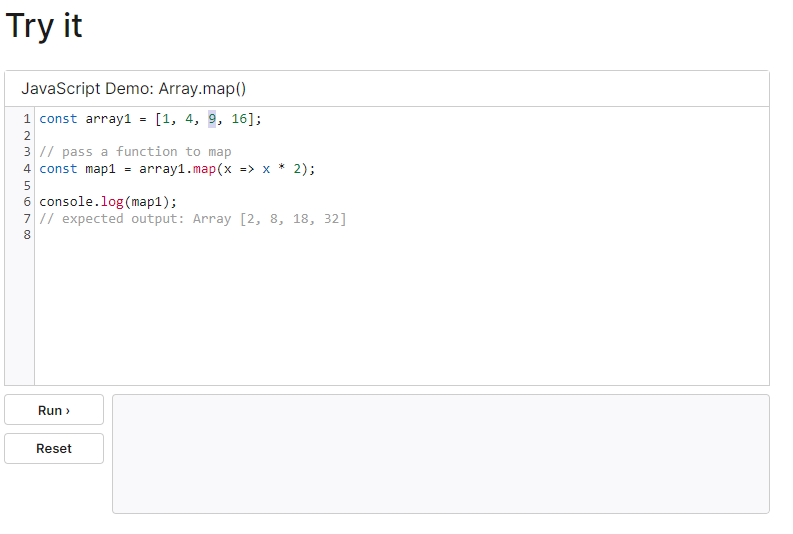
takes a function, which we pass as an argument, and that function is then executed

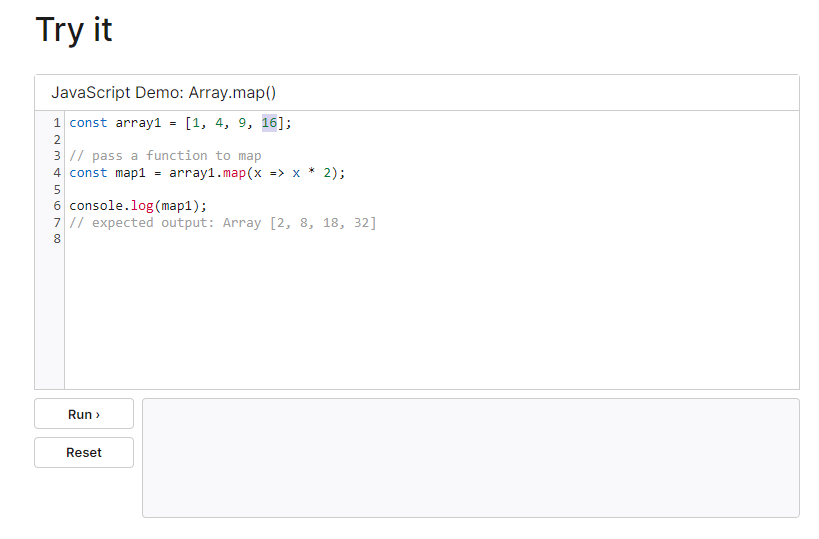


for every item in the array on which we're





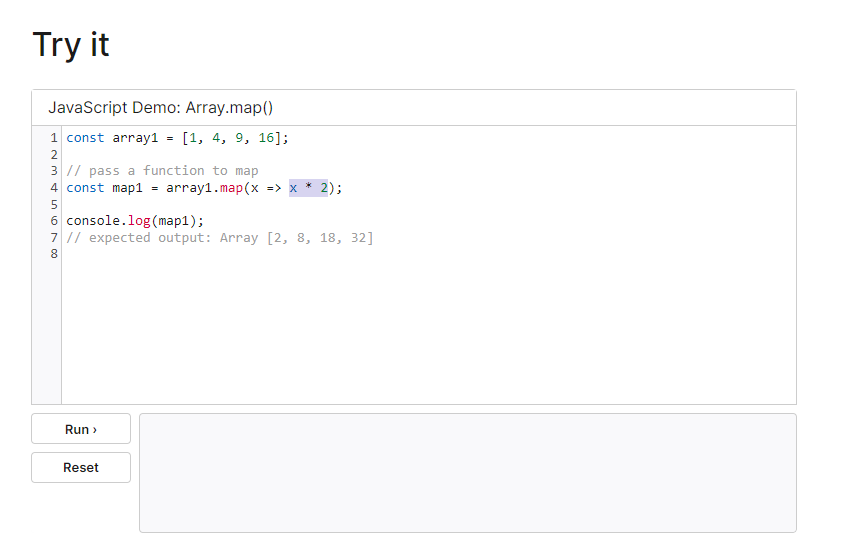




calling map(),



and the result of this function is the element which will be added to the newly created array.



That's how the map() method works. It's a built-in method, and we can utilize it in the Expenses component to transform our array of objects, which we have here,

|  |
| --- |
| src/App.js |
| import React from "react";  import NewExpense from "./components/NewExpense/NewExpense";  import Expenses from "./components/Expenses/Expenses";  const App = () => {  const expenses = **[**  **{**  **id: "e1",**  **title: "Toilet Paper",**  **amount: 94.12,**  **date: new Date(2020, 7, 14),**  **},**  **{ id: "e2", title: "New TV", amount: 799.49, date: new Date(2021, 2, 12) },**  **{**  **id: "e3",**  **title: "Car Insurance",**  **amount: 294.67,**  **date: new Date(2021, 2, 28),**  **},**  **{**  **id: "e4",**  **title: "New Desk (Wooden)",**  **amount: 450,**  **date: new Date(2021, 5, 12),**  **},**  **]**;  const addExpenseHandler = expense => {  console.log('In App.js');  console.log(expense);  }  return (  <div>  <NewExpense onAddExpense={addExpenseHandler}/>  <Expenses items={expenses} />  </div>  );  };  export default App; |

into an array of JSX elements, to be precise into an array full of ExpenseItem JSX elements.

|  |
| --- |
| src/components/Expenses/Expenses.js |
| import React, {useState} from "react";  import ExpenseItem from "./ExpenseItem";  import ExpensesFilter from "./ExpensesFilter";  import Card from "../UI/Card";  import "./Expenses.css";  const Expenses = (props) => {  const [filteredYear, setFilteredYear] = useState('2020');  const filterChangeHandler = selectedYear => {  setFilteredYear(selectedYear);  }  return (  <div>  <Card className="expenses">  <ExpensesFilter selected={filteredYear} onChangeFilter={filterChangeHandler}/>  {props.items.map()}  **<ExpenseItem**  **title={props.items[0].title}**  **amount={props.items[0].amount}**  **date={props.items[0].date}**  **/>**  <ExpenseItem  title={props.items[1].title}  amount={props.items[1].amount}  date={props.items[1].date}  />  <ExpenseItem  title={props.items[2].title}  amount={props.items[2].amount}  date={props.items[2].date}  />  <ExpenseItem  title={props.items[3].title}  amount={props.items[3].amount}  date={props.items[3].date}  />  </Card>  </div>  );  };  export default Expenses; |

If you do output an array of JSX elements here,

|  |
| --- |
| src/components/Expenses/Expenses.js |
| import React, {useState} from "react";  import ExpenseItem from "./ExpenseItem";  import ExpensesFilter from "./ExpensesFilter";  import Card from "../UI/Card";  import "./Expenses.css";  const Expenses = (props) => {  const [filteredYear, setFilteredYear] = useState('2020');  const filterChangeHandler = selectedYear => {  setFilteredYear(selectedYear);  }  return (  <div>  <Card className="expenses">  <ExpensesFilter selected={filteredYear} onChangeFilter={filterChangeHandler}/>  {props.items.map()}  **{}**  <ExpenseItem  title={props.items[0].title}  amount={props.items[0].amount}  date={props.items[0].date}  />  <ExpenseItem  title={props.items[1].title}  amount={props.items[1].amount}  date={props.items[1].date}  />  <ExpenseItem  title={props.items[2].title}  amount={props.items[2].amount}  date={props.items[2].date}  />  <ExpenseItem  title={props.items[3].title}  amount={props.items[3].amount}  date={props.items[3].date}  />  </Card>  </div>  );  };  export default Expenses; |

like a couple of Cards, then React is capable of simply rendering these elements. If you had something like this, an array of JSX elements as part of your JSX code, React would simply render these elements side by side.

|  |
| --- |
| src/components/Expenses/Expenses.js |
| import React, {useState} from "react";  import ExpenseItem from "./ExpenseItem";  import ExpensesFilter from "./ExpensesFilter";  import Card from "../UI/Card";  import "./Expenses.css";  const Expenses = (props) => {  const [filteredYear, setFilteredYear] = useState('2020');  const filterChangeHandler = selectedYear => {  setFilteredYear(selectedYear);  }  return (  <div>  <Card className="expenses">  <ExpensesFilter selected={filteredYear} onChangeFilter={filterChangeHandler}/>  {props.items.map()}  **{[<Card/>, <Card/>]}**  <ExpenseItem  title={props.items[0].title}  amount={props.items[0].amount}  date={props.items[0].date}  />  <ExpenseItem  title={props.items[1].title}  amount={props.items[1].amount}  date={props.items[1].date}  />  <ExpenseItem  title={props.items[2].title}  amount={props.items[2].amount}  date={props.items[2].date}  />  <ExpenseItem  title={props.items[3].title}  amount={props.items[3].amount}  date={props.items[3].date}  />  </Card>  </div>  );  };  export default Expenses; |

And that's what we utilize here to transform our array of objects into such an array of ExpenseItems, which then are rendered by React.

|  |
| --- |
| src/components/Expenses/Expenses.js |
| import React, {useState} from "react";  import ExpenseItem from "./ExpenseItem";  import ExpensesFilter from "./ExpensesFilter";  import Card from "../UI/Card";  import "./Expenses.css";  const Expenses = (props) => {  const [filteredYear, setFilteredYear] = useState('2020');  const filterChangeHandler = selectedYear => {  setFilteredYear(selectedYear);  }  return (  <div>  <Card className="expenses">  <ExpensesFilter selected={filteredYear} onChangeFilter={filterChangeHandler}/>  {props.items.map**()**}  <ExpenseItem  title={props.items[0].title}  amount={props.items[0].amount}  date={props.items[0].date}  />  <ExpenseItem  title={props.items[1].title}  amount={props.items[1].amount}  date={props.items[1].date}  />  <ExpenseItem  title={props.items[2].title}  amount={props.items[2].amount}  date={props.items[2].date}  />  <ExpenseItem  title={props.items[3].title}  amount={props.items[3].amount}  date={props.items[3].date}  />  </Card>  </div>  );  };  export default Expenses; |

So, map takes a function as an argument, and that function executes for every element in the array, and it gets that element for which it's currently executing as a parameter. So here we get our expense, for example, and then

|  |
| --- |
| src/components/Expenses/Expenses.js |
| import React, {useState} from "react";  import ExpenseItem from "./ExpenseItem";  import ExpensesFilter from "./ExpensesFilter";  import Card from "../UI/Card";  import "./Expenses.css";  const Expenses = (props) => {  const [filteredYear, setFilteredYear] = useState('2020');  const filterChangeHandler = selectedYear => {  setFilteredYear(selectedYear);  }  return (  <div>  <Card className="expenses">  <ExpensesFilter selected={filteredYear} onChangeFilter={filterChangeHandler}/>  {props.items.map(**expense** => )}  <ExpenseItem  title={props.items[0].title}  amount={props.items[0].amount}  date={props.items[0].date}  />  <ExpenseItem  title={props.items[1].title}  amount={props.items[1].amount}  date={props.items[1].date}  />  <ExpenseItem  title={props.items[2].title}  amount={props.items[2].amount}  date={props.items[2].date}  />  <ExpenseItem  title={props.items[3].title}  amount={props.items[3].amount}  date={props.items[3].date}  />  </Card>  </div>  );  };  export default Expenses; |

in the function body, so on the right side of the arrow, we then have to return the JSX element into which we want to map this expense. So, in this case, I want to map every expense in my expenses array into an ExpenseItem JSX element.

|  |
| --- |
| src/components/Expenses/Expenses.js |
| import React, {useState} from "react";  import ExpenseItem from "./ExpenseItem";  import ExpensesFilter from "./ExpensesFilter";  import Card from "../UI/Card";  import "./Expenses.css";  const Expenses = (props) => {  const [filteredYear, setFilteredYear] = useState('2020');  const filterChangeHandler = selectedYear => {  setFilteredYear(selectedYear);  }  return (  <div>  <Card className="expenses">  <ExpensesFilter selected={filteredYear} onChangeFilter={filterChangeHandler}/>  {props.items.map(expense => **<ExpenseItem />**)}  <ExpenseItem  title={props.items[0].title}  amount={props.items[0].amount}  date={props.items[0].date}  />  <ExpenseItem  title={props.items[1].title}  amount={props.items[1].amount}  date={props.items[1].date}  />  <ExpenseItem  title={props.items[2].title}  amount={props.items[2].amount}  date={props.items[2].date}  />  <ExpenseItem  title={props.items[3].title}  amount={props.items[3].amount}  date={props.items[3].date}  />  </Card>  </div>  );  };  export default Expenses; |

So I want to transform the expense object,

|  |
| --- |
| src/components/Expenses/Expenses.js |
| import React, {useState} from "react";  import ExpenseItem from "./ExpenseItem";  import ExpensesFilter from "./ExpensesFilter";  import Card from "../UI/Card";  import "./Expenses.css";  const Expenses = (props) => {  const [filteredYear, setFilteredYear] = useState('2020');  const filterChangeHandler = selectedYear => {  setFilteredYear(selectedYear);  }  return (  <div>  <Card className="expenses">  <ExpensesFilter selected={filteredYear} onChangeFilter={filterChangeHandler}/>  {props.items.map(**expense** => <ExpenseItem />)}  <ExpenseItem  title={props.items[0].title}  amount={props.items[0].amount}  date={props.items[0].date}  />  <ExpenseItem  title={props.items[1].title}  amount={props.items[1].amount}  date={props.items[1].date}  />  <ExpenseItem  title={props.items[2].title}  amount={props.items[2].amount}  date={props.items[2].date}  />  <ExpenseItem  title={props.items[3].title}  amount={props.items[3].amount}  date={props.items[3].date}  />  </Card>  </div>  );  };  export default Expenses; |

to this special kind of object, to this JSX element,

|  |
| --- |
| src/components/Expenses/Expenses.js |
| import React, {useState} from "react";  import ExpenseItem from "./ExpenseItem";  import ExpensesFilter from "./ExpensesFilter";  import Card from "../UI/Card";  import "./Expenses.css";  const Expenses = (props) => {  const [filteredYear, setFilteredYear] = useState('2020');  const filterChangeHandler = selectedYear => {  setFilteredYear(selectedYear);  }  return (  <div>  <Card className="expenses">  <ExpensesFilter selected={filteredYear} onChangeFilter={filterChangeHandler}/>  {props.items.map(expense => **<ExpenseItem />**)}  <ExpenseItem  title={props.items[0].title}  amount={props.items[0].amount}  date={props.items[0].date}  />  <ExpenseItem  title={props.items[1].title}  amount={props.items[1].amount}  date={props.items[1].date}  />  <ExpenseItem  title={props.items[2].title}  amount={props.items[2].amount}  date={props.items[2].date}  />  <ExpenseItem  title={props.items[3].title}  amount={props.items[3].amount}  date={props.items[3].date}  />  </Card>  </div>  );  };  export default Expenses; |

and then we can just configure this as we did it here.

|  |
| --- |
| src/components/Expenses/Expenses.js |
| import React, {useState} from "react";  import ExpenseItem from "./ExpenseItem";  import ExpensesFilter from "./ExpensesFilter";  import Card from "../UI/Card";  import "./Expenses.css";  const Expenses = (props) => {  const [filteredYear, setFilteredYear] = useState('2020');  const filterChangeHandler = selectedYear => {  setFilteredYear(selectedYear);  }  return (  <div>  <Card className="expenses">  <ExpensesFilter selected={filteredYear} onChangeFilter={filterChangeHandler}/>  {props.items.map(expense => <ExpenseItem />)}  **<ExpenseItem**  **title={props.items[0].title}**  **amount={props.items[0].amount}**  **date={props.items[0].date}**  **/>**  <ExpenseItem  title={props.items[1].title}  amount={props.items[1].amount}  date={props.items[1].date}  />  <ExpenseItem  title={props.items[2].title}  amount={props.items[2].amount}  date={props.items[2].date}  />  <ExpenseItem  title={props.items[3].title}  amount={props.items[3].amount}  date={props.items[3].date}  />  </Card>  </div>  );  };  export default Expenses; |

We add a title prop, but the title is now expense.title

|  |
| --- |
| src/components/Expenses/Expenses.js |
| import React, {useState} from "react";  import ExpenseItem from "./ExpenseItem";  import ExpensesFilter from "./ExpensesFilter";  import Card from "../UI/Card";  import "./Expenses.css";  const Expenses = (props) => {  const [filteredYear, setFilteredYear] = useState('2020');  const filterChangeHandler = selectedYear => {  setFilteredYear(selectedYear);  }  return (  <div>  <Card className="expenses">  <ExpensesFilter selected={filteredYear} onChangeFilter={filterChangeHandler}/>  {props.items.map(expense => <ExpenseItem **title={expense.title}** />)}  <ExpenseItem  title={props.items[0].title}  amount={props.items[0].amount}  date={props.items[0].date}  />  <ExpenseItem  title={props.items[1].title}  amount={props.items[1].amount}  date={props.items[1].date}  />  <ExpenseItem  title={props.items[2].title}  amount={props.items[2].amount}  date={props.items[2].date}  />  <ExpenseItem  title={props.items[3].title}  amount={props.items[3].amount}  date={props.items[3].date}  />  </Card>  </div>  );  };  export default Expenses; |

So, this expense, which is passed as a parameter

|  |
| --- |
| src/components/Expenses/Expenses.js |
| import React, {useState} from "react";  import ExpenseItem from "./ExpenseItem";  import ExpensesFilter from "./ExpensesFilter";  import Card from "../UI/Card";  import "./Expenses.css";  const Expenses = (props) => {  const [filteredYear, setFilteredYear] = useState('2020');  const filterChangeHandler = selectedYear => {  setFilteredYear(selectedYear);  }  return (  <div>  <Card className="expenses">  <ExpensesFilter selected={filteredYear} onChangeFilter={filterChangeHandler}/>  {props.items.map(**expense** => <ExpenseItem title={expense.title} />)}  <ExpenseItem  title={props.items[0].title}  amount={props.items[0].amount}  date={props.items[0].date}  />  <ExpenseItem  title={props.items[1].title}  amount={props.items[1].amount}  date={props.items[1].date}  />  <ExpenseItem  title={props.items[2].title}  amount={props.items[2].amount}  date={props.items[2].date}  />  <ExpenseItem  title={props.items[3].title}  amount={props.items[3].amount}  date={props.items[3].date}  />  </Card>  </div>  );  };  export default Expenses; |

into this function automatically, because that's how map works,

|  |
| --- |
| src/components/Expenses/Expenses.js |
| import React, {useState} from "react";  import ExpenseItem from "./ExpenseItem";  import ExpensesFilter from "./ExpensesFilter";  import Card from "../UI/Card";  import "./Expenses.css";  const Expenses = (props) => {  const [filteredYear, setFilteredYear] = useState('2020');  const filterChangeHandler = selectedYear => {  setFilteredYear(selectedYear);  }  return (  <div>  <Card className="expenses">  <ExpensesFilter selected={filteredYear} onChangeFilter={filterChangeHandler}/>  {props.items.map(**expense => <ExpenseItem title={expense.title} />**)}  <ExpenseItem  title={props.items[0].title}  amount={props.items[0].amount}  date={props.items[0].date}  />  <ExpenseItem  title={props.items[1].title}  amount={props.items[1].amount}  date={props.items[1].date}  />  <ExpenseItem  title={props.items[2].title}  amount={props.items[2].amount}  date={props.items[2].date}  />  <ExpenseItem  title={props.items[3].title}  amount={props.items[3].amount}  date={props.items[3].date}  />  </Card>  </div>  );  };  export default Expenses; |

that expense is used

|  |
| --- |
| src/components/Expenses/Expenses.js |
| import React, {useState} from "react";  import ExpenseItem from "./ExpenseItem";  import ExpensesFilter from "./ExpensesFilter";  import Card from "../UI/Card";  import "./Expenses.css";  const Expenses = (props) => {  const [filteredYear, setFilteredYear] = useState('2020');  const filterChangeHandler = selectedYear => {  setFilteredYear(selectedYear);  }  return (  <div>  <Card className="expenses">  <ExpensesFilter selected={filteredYear} onChangeFilter={filterChangeHandler}/>  {props.items.map(**expense** => <ExpenseItem title={expense.title} />)}  <ExpenseItem  title={props.items[0].title}  amount={props.items[0].amount}  date={props.items[0].date}  />  <ExpenseItem  title={props.items[1].title}  amount={props.items[1].amount}  date={props.items[1].date}  />  <ExpenseItem  title={props.items[2].title}  amount={props.items[2].amount}  date={props.items[2].date}  />  <ExpenseItem  title={props.items[3].title}  amount={props.items[3].amount}  date={props.items[3].date}  />  </Card>  </div>  );  };  export default Expenses; |

to extract the title.

|  |
| --- |
| src/components/Expenses/Expenses.js |
| import React, {useState} from "react";  import ExpenseItem from "./ExpenseItem";  import ExpensesFilter from "./ExpensesFilter";  import Card from "../UI/Card";  import "./Expenses.css";  const Expenses = (props) => {  const [filteredYear, setFilteredYear] = useState('2020');  const filterChangeHandler = selectedYear => {  setFilteredYear(selectedYear);  }  return (  <div>  <Card className="expenses">  <ExpensesFilter selected={filteredYear} onChangeFilter={filterChangeHandler}/>  {props.items.map(expense => <ExpenseItem title={**expense.title**} />)}  <ExpenseItem  title={props.items[0].title}  amount={props.items[0].amount}  date={props.items[0].date}  />  <ExpenseItem  title={props.items[1].title}  amount={props.items[1].amount}  date={props.items[1].date}  />  <ExpenseItem  title={props.items[2].title}  amount={props.items[2].amount}  date={props.items[2].date}  />  <ExpenseItem  title={props.items[3].title}  amount={props.items[3].amount}  date={props.items[3].date}  />  </Card>  </div>  );  };  export default Expenses; |

We can also use expense to extract the amount and date.

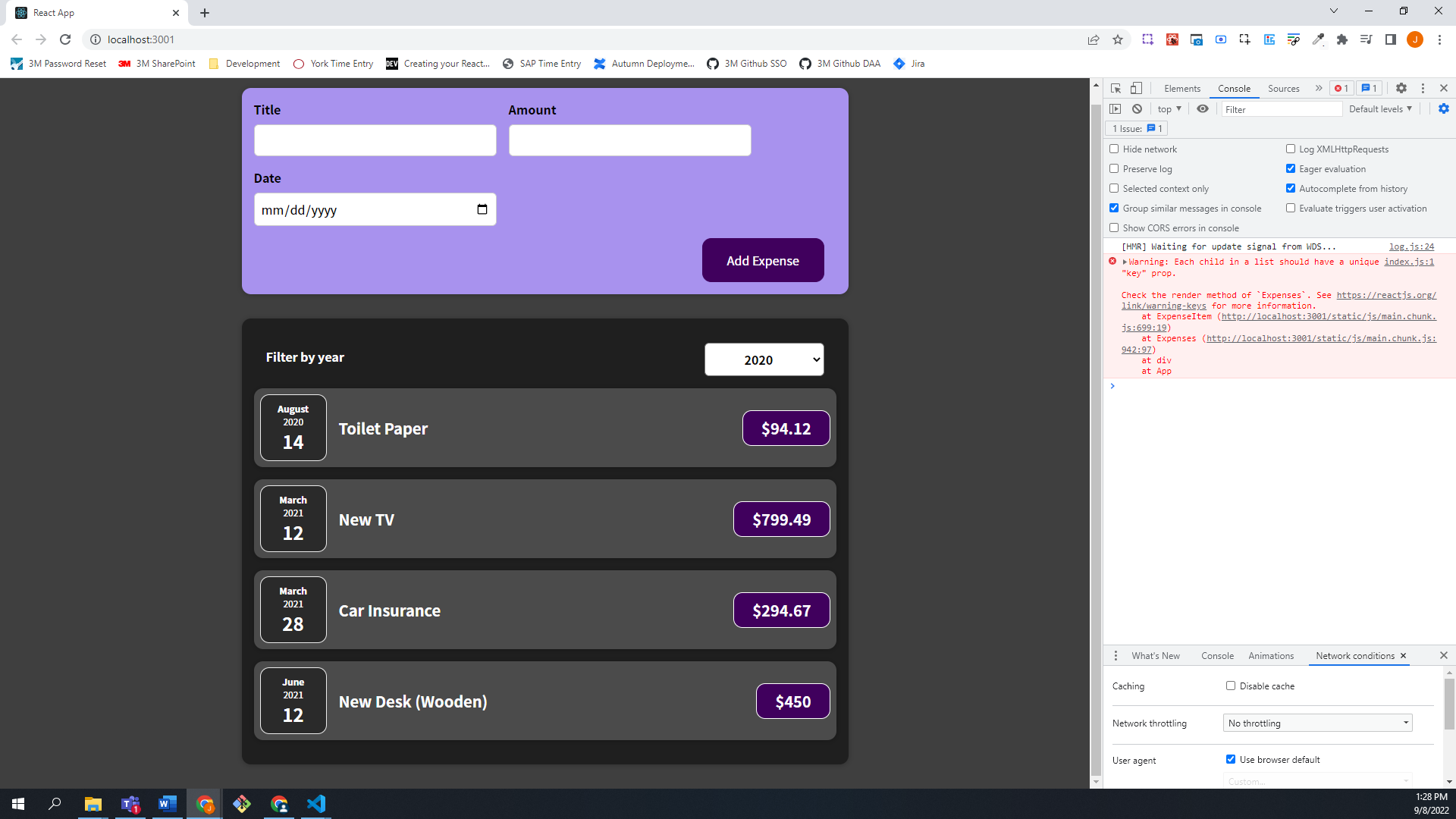
|  |
| --- |
| src/components/Expenses/Expenses.js |
| import React, { useState } from "react";  import ExpenseItem from "./ExpenseItem";  import ExpensesFilter from "./ExpensesFilter";  import Card from "../UI/Card";  import "./Expenses.css";  const Expenses = (props) => {  const [filteredYear, setFilteredYear] = useState("2020");  const filterChangeHandler = (selectedYear) => {  setFilteredYear(selectedYear);  };  return (  <div>  <Card className="expenses">  <ExpensesFilter  selected={filteredYear}  onChangeFilter={filterChangeHandler}  />  {props.items.map((expense) => (  <ExpenseItem  title={expense.title}  **amount={expense.amount}**  **date={expense.date}**  />  ))}  <ExpenseItem  title={props.items[0].title}  amount={props.items[0].amount}  date={props.items[0].date}  />  <ExpenseItem  title={props.items[1].title}  amount={props.items[1].amount}  date={props.items[1].date}  />  <ExpenseItem  title={props.items[2].title}  amount={props.items[2].amount}  date={props.items[2].date}  />  <ExpenseItem  title={props.items[3].title}  amount={props.items[3].amount}  date={props.items[3].date}  />  </Card>  </div>  );  };  export default Expenses; |

Now, we can get rid of the hard-coded ExpenseItems here.

|  |
| --- |
| src/components/Expenses/Expenses.js |
| import React, { useState } from "react";  import ExpenseItem from "./ExpenseItem";  import ExpensesFilter from "./ExpensesFilter";  import Card from "../UI/Card";  import "./Expenses.css";  const Expenses = (props) => {  const [filteredYear, setFilteredYear] = useState("2020");  const filterChangeHandler = (selectedYear) => {  setFilteredYear(selectedYear);  };  return (  <div>  <Card className="expenses">  <ExpensesFilter  selected={filteredYear}  onChangeFilter={filterChangeHandler}  />  {props.items.map((expense) => (  <ExpenseItem  title={expense.title}  amount={expense.amount}  date={expense.date}  />  ))}  ~~<ExpenseItem~~  ~~title={props.items[0].title}~~  ~~amount={props.items[0].amount}~~  ~~date={props.items[0].date}~~  ~~/>~~  ~~<ExpenseItem~~  ~~title={props.items[1].title}~~  ~~amount={props.items[1].amount}~~  ~~date={props.items[1].date}~~  ~~/>~~  ~~<ExpenseItem~~  ~~title={props.items[2].title}~~  ~~amount={props.items[2].amount}~~  ~~date={props.items[2].date}~~  ~~/>~~  ~~<ExpenseItem~~  ~~title={props.items[3].title}~~  ~~amount={props.items[3].amount}~~  ~~date={props.items[3].date}~~  ~~/>~~  </Card>  </div>  );  };  export default Expenses; |

|  |
| --- |
| src/components/Expenses/Expenses.js |
| import React, { useState } from "react";  import ExpenseItem from "./ExpenseItem";  import ExpensesFilter from "./ExpensesFilter";  import Card from "../UI/Card";  import "./Expenses.css";  const Expenses = (props) => {  const [filteredYear, setFilteredYear] = useState("2020");  const filterChangeHandler = (selectedYear) => {  setFilteredYear(selectedYear);  };  return (  <div>  <Card className="expenses">  <ExpensesFilter  selected={filteredYear}  onChangeFilter={filterChangeHandler}  />  {props.items.map((expense) => (  <ExpenseItem  title={expense.title}  amount={expense.amount}  date={expense.date}  />  ))}  </Card>  </div>  );  };  export default Expenses; |

If we now save this and reload the webpage, we get a warning, which we can ignore for now. We also see that we get all these items being rendered just as before but now it's actually happening dynamically, which simply means that now it's based on the actual array. This means that we can now also change the array and such changes will be reflected in that list, which is going to be what's talked about in the next lecture.



### 65. Using Stateful Lists

How can we now update this expenses array whenever a new expense is being added? For this we need to go to App.js because that is where we have the expenses array as well as the function addExpenseHandler, which is triggered whenever a new expense is added.

What we have to do is add this expense to

|  |
| --- |
| src/App.js |
| import React from "react";  import NewExpense from "./components/NewExpense/NewExpense";  import Expenses from "./components/Expenses/Expenses";  const App = () => {  const expenses = [  {  id: "e1",  title: "Toilet Paper",  amount: 94.12,  date: new Date(2020, 7, 14),  },  { id: "e2", title: "New TV", amount: 799.49, date: new Date(2021, 2, 12) },  {  id: "e3",  title: "Car Insurance",  amount: 294.67,  date: new Date(2021, 2, 28),  },  {  id: "e4",  title: "New Desk (Wooden)",  amount: 450,  date: new Date(2021, 5, 12),  },  ];  const addExpenseHandler = **expense** => {  console.log('In App.js');  console.log(expense);  }  return (  <div>  <NewExpense onAddExpense={addExpenseHandler}/>  <Expenses items={expenses} />  </div>  );  };  export default App; |

this expenses array. However, if we just add it, React won't update the component. Instead, to get React to update the component, you need to use state.

|  |
| --- |
| src/App.js |
| import React from "react";  import NewExpense from "./components/NewExpense/NewExpense";  import Expenses from "./components/Expenses/Expenses";  const App = () => {  **const expenses = [**  **{**  **id: "e1",**  **title: "Toilet Paper",**  **amount: 94.12,**  **date: new Date(2020, 7, 14),**  **},**  **{ id: "e2", title: "New TV", amount: 799.49, date: new Date(2021, 2, 12) },**  **{**  **id: "e3",**  **title: "Car Insurance",**  **amount: 294.67,**  **date: new Date(2021, 2, 28),**  **},**  **{**  **id: "e4",**  **title: "New Desk (Wooden)",**  **amount: 450,**  **date: new Date(2021, 5, 12),**  **},**  **];**  const addExpenseHandler = expense => {  console.log('In App.js');  console.log(expense);  }  return (  <div>  <NewExpense onAddExpense={addExpenseHandler}/>  <Expenses items={expenses} />  </div>  );  };  export default App; |

Now in App.js we want to import useState.

|  |
| --- |
| src/App.js |
| import React, **{useState}** from "react";  import NewExpense from "./components/NewExpense/NewExpense";  import Expenses from "./components/Expenses/Expenses";  const App = () => {  const expenses = [  {  id: "e1",  title: "Toilet Paper",  amount: 94.12,  date: new Date(2020, 7, 14),  },  { id: "e2", title: "New TV", amount: 799.49, date: new Date(2021, 2, 12) },  {  id: "e3",  title: "Car Insurance",  amount: 294.67,  date: new Date(2021, 2, 28),  },  {  id: "e4",  title: "New Desk (Wooden)",  amount: 450,  date: new Date(2021, 5, 12),  },  ];  const addExpenseHandler = expense => {  console.log('In App.js');  console.log(expense);  }  return (  <div>  <NewExpense onAddExpense={addExpenseHandler}/>  <Expenses items={expenses} />  </div>  );  };  export default App; |

We will actually grab the expenses array, extract it out of the App component function, store it above the App component function declaration, and name it DUMMY\_EXPENSES, as this is just some dummy data that we can use to initialize our state with.

|  |
| --- |
| src/App.js |
| import React, {useState} from "react";  import NewExpense from "./components/NewExpense/NewExpense";  import Expenses from "./components/Expenses/Expenses";  **const DUMMY\_EXPENSES = [**  **{**  **id: "e1",**  **title: "Toilet Paper",**  **amount: 94.12,**  **date: new Date(2020, 7, 14),**  **},**  **{ id: "e2", title: "New TV", amount: 799.49, date: new Date(2021, 2, 12) },**  **{**  **id: "e3",**  **title: "Car Insurance",**  **amount: 294.67,**  **date: new Date(2021, 2, 28),**  **},**  **{**  **id: "e4",**  **title: "New Desk (Wooden)",**  **amount: 450,**  **date: new Date(2021, 5, 12),**  **},**  **];**  const App = () => {  const addExpenseHandler = expense => {  console.log('In App.js');  console.log(expense);  }  return (  <div>  <NewExpense onAddExpense={addExpenseHandler}/>  <Expenses items={expenses} />  </div>  );  };  export default App; |

Inside of the App component function, we call useState, and I'll pass DUMMY\_EXPENSES as an initial state value, so that we have some initial expenses to display. We can then use destructuring to get access to the expenses and to the setExpenses state updating function.

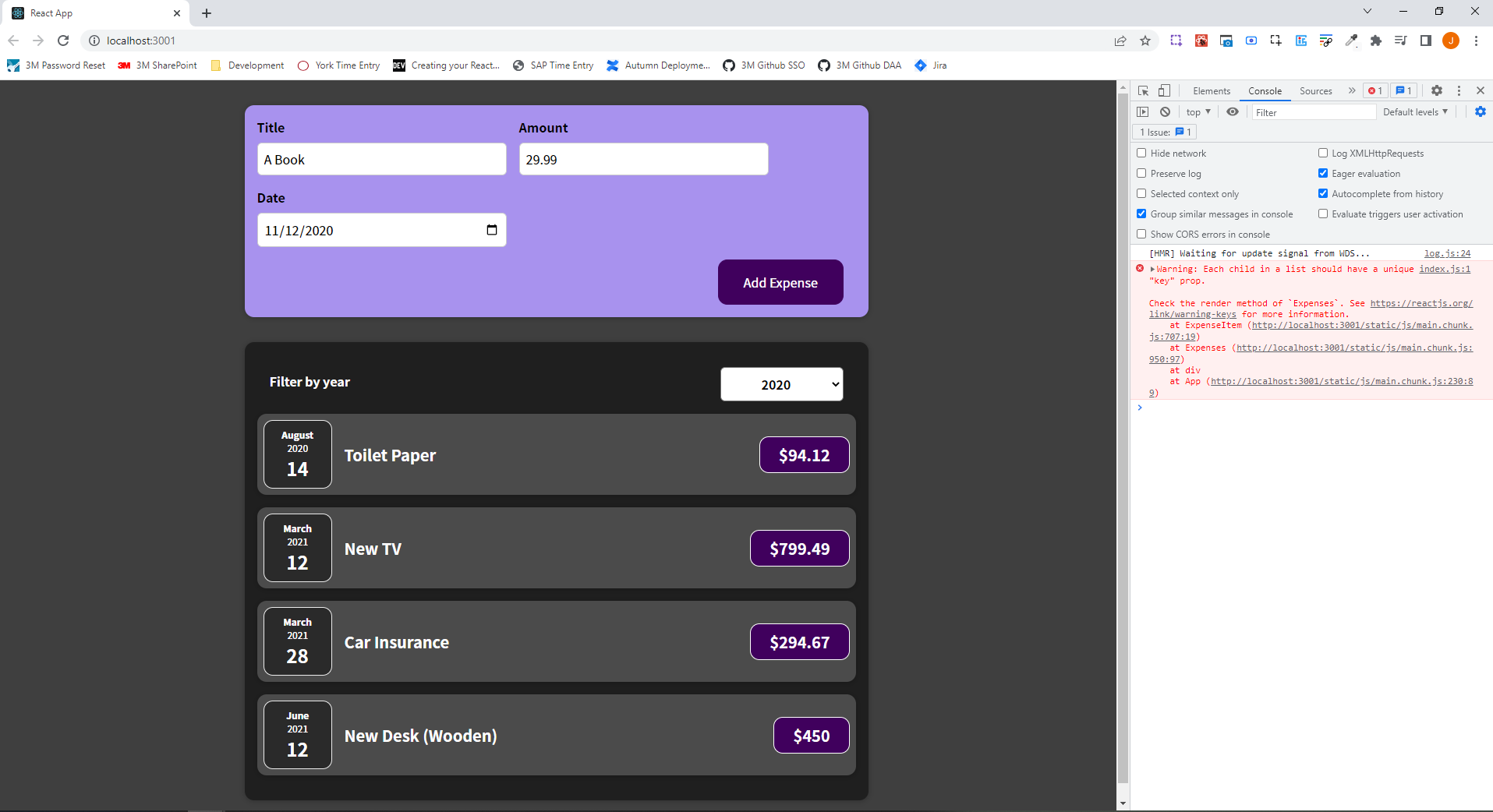
|  |
| --- |
| src/App.js |
| import React, {useState} from "react";  import NewExpense from "./components/NewExpense/NewExpense";  import Expenses from "./components/Expenses/Expenses";  const DUMMY\_EXPENSES = [  {  id: "e1",  title: "Toilet Paper",  amount: 94.12,  date: new Date(2020, 7, 14),  },  { id: "e2", title: "New TV", amount: 799.49, date: new Date(2021, 2, 12) },  {  id: "e3",  title: "Car Insurance",  amount: 294.67,  date: new Date(2021, 2, 28),  },  {  id: "e4",  title: "New Desk (Wooden)",  amount: 450,  date: new Date(2021, 5, 12),  },  ];  const App = () => {  **const [expenses, setExpenses] = useState(DUMMY\_EXPENSES);**  const addExpenseHandler = expense => {  console.log('In App.js');  console.log(expense);  }  return (  <div>  <NewExpense onAddExpense={addExpenseHandler}/>  <Expenses items={expenses} />  </div>  );  };  export default App; |

In the function addExpenseHandler, we can now remove the console.log() statements. In the function addExpenseHandler, where we want to add a new expense, we can then call setExpenses to set our expenses array to a new array which includes the new expense. Inside of setExpenses, we will update the state based on the previous state, or the previous snapshot of this state. As we learned, we should use the special function form for this state updating function, so it would be passed a function as an argument to this state updating function, and that function will automatically receive the latest state snapshot. So here we would then get our prevExpenses automatically by react and we would return our new array where we add this expense, which we're getting as a parameter here, and where we then add our prevExpenses with the spread operator.

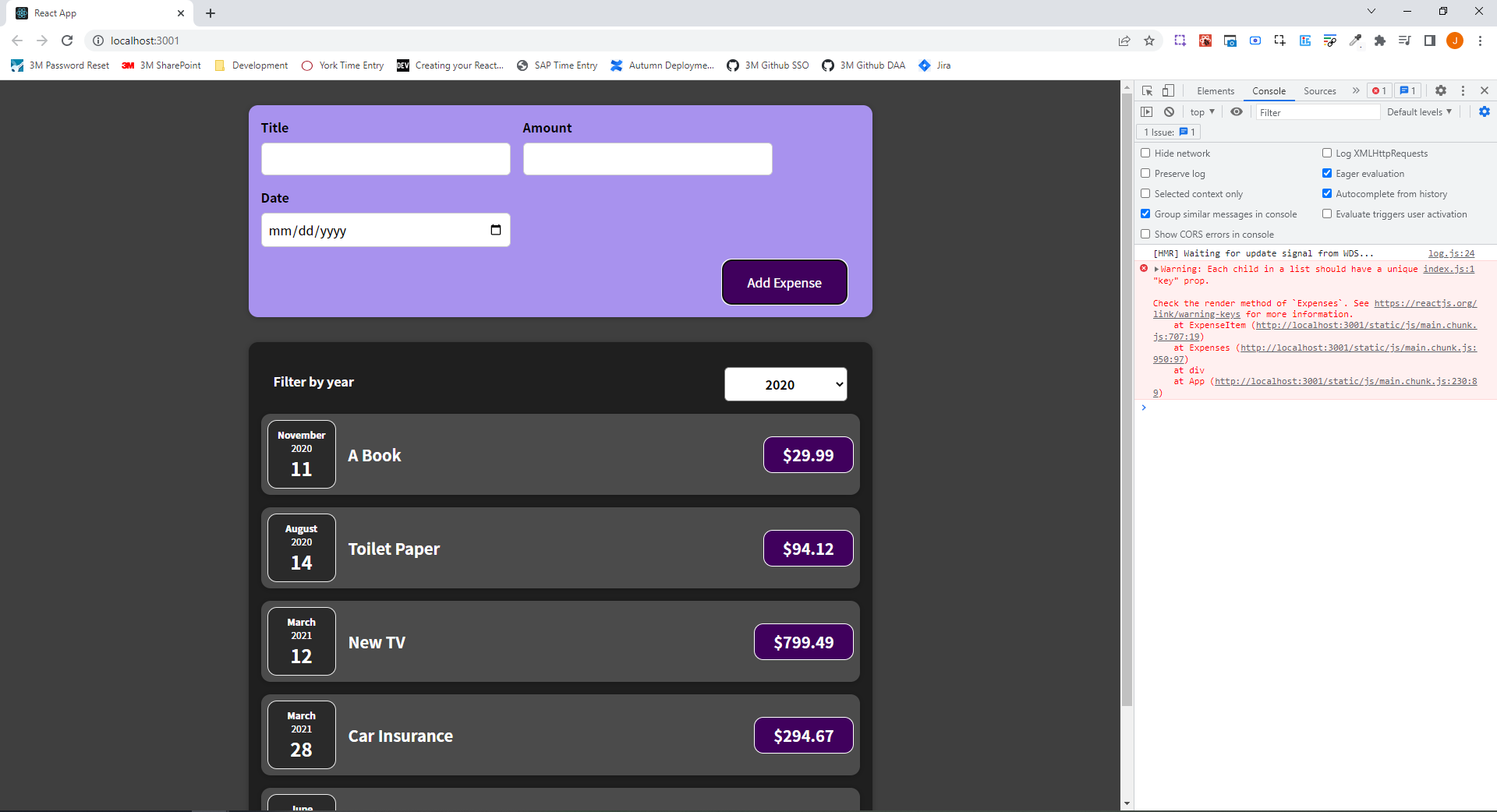
|  |
| --- |
| src/App.js |
| import React, {useState} from "react";  import NewExpense from "./components/NewExpense/NewExpense";  import Expenses from "./components/Expenses/Expenses";  const DUMMY\_EXPENSES = [  {  id: "e1",  title: "Toilet Paper",  amount: 94.12,  date: new Date(2020, 7, 14),  },  { id: "e2", title: "New TV", amount: 799.49, date: new Date(2021, 2, 12) },  {  id: "e3",  title: "Car Insurance",  amount: 294.67,  date: new Date(2021, 2, 28),  },  {  id: "e4",  title: "New Desk (Wooden)",  amount: 450,  date: new Date(2021, 5, 12),  },  ];  const App = () => {  const [expenses, setExpenses] = useState(DUMMY\_EXPENSES);  const addExpenseHandler = expense => {  **setExpenses((prevExpenses) => {**  **return [expense, ...prevExpenses];**  **});**  }  return (  <div>  <NewExpense onAddExpense={addExpenseHandler}/>  <Expenses items={expenses } />  </div>  );  };  export default App; |

We should now have a dynamic list, which automatically updates when items are added.

So now in the browser, if we refresh the page and we now add a book for $29.99, pick some date, and click "Add Expense",

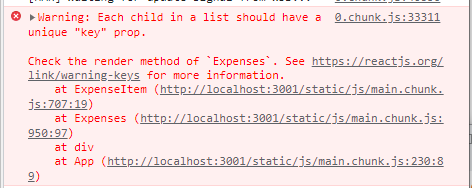


We will see A Book added as the first item. So now this is working and is truly a dynamic list by using state and by outputting the items using the map()method.



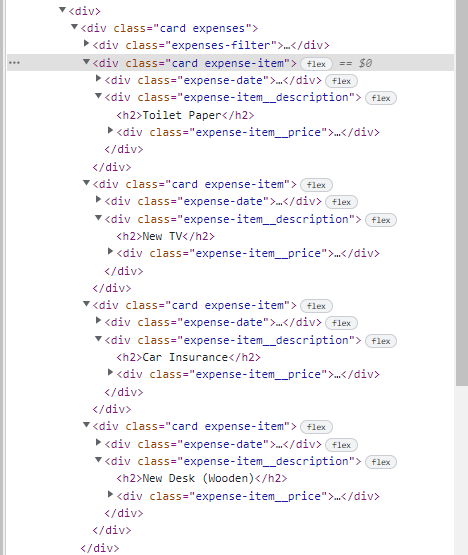
### 66. Understanding "Keys"

Why are we getting this "Key" warning? It looks like everything works.



Well React has a special concept when it comes to rendering lists of data, a concept which exists to ensure that React is able to update and render such lists efficiently without performance losses, or bugs, which may occur.

If we select the Elements tab in the Developer Tools, and hover over the list of items, we will see multiple divs representing each ExpenseItem.

****

If we add a new item, React renders this new item as the last item in the list of div's and updates all items and replaces their content, such that it again matches the order of the items in my Array. This is not great. This is happening because to React all of these items look similar, and it only sees that my Array changed; that my array is now longer than before. And, hence, it simply renders an additional div, and it adds that at the end. It then walks through all of the items and updates the content inside of every item to match the Array content again. The final result is correct but from a performance perspective, this is not great because all items are visited and updated, and it can even lead to bugs.

If the ExpenseItems would be stateful items and we would have some state managed inside of that, then if, for example, our first item, if it has a certain state, if we add a new item, the old first item would be overwritten with the new first item. Hence, any state changes we might’ve had in there would be lost. So besides potential performance issues, we could also run into bugs.

We have a way of telling React where a new item should be added. We do that by going to the place where we output our list of items,

|  |
| --- |
| src/components/Expenses/Expenses.js |
| import React, { useState } from "react";  import ExpenseItem from "./ExpenseItem";  import ExpensesFilter from "./ExpensesFilter";  import Card from "../UI/Card";  import "./Expenses.css";  const Expenses = (props) => {  const [filteredYear, setFilteredYear] = useState("2020");  const filterChangeHandler = (selectedYear) => {  setFilteredYear(selectedYear);  };  return (  <div>  <Card className="expenses">  <ExpensesFilter  selected={filteredYear}  onChangeFilter={filterChangeHandler}  />  **{props.items.map((expense) => (**  **<ExpenseItem**  **title={expense.title}**  **amount={expense.amount}**  **date={expense.date}**  **/>**  **))}**  </Card>  </div>  );  };  export default Expenses; |

We then add a special prop to this ExpenseItem here, and that's the key prop. The key prop is not a prop that we are using inside of ExpenseItem, but instead it's a prop you can add to any component no matter if it's a custom component by you or it's a built-in HTML element, you can always add this. And if you add the key prop, then you can help React identify the individual items. To identify the individual items, you need to set a unique value per list item, and that for example in this case here, would be expense.id because in the array called DUMMY\_EXPENSES holding our initial expenses, every item has a unique ID.

|  |
| --- |
| src/components/Expenses/Expenses.js |
| import React, { useState } from "react";  import ExpenseItem from "./ExpenseItem";  import ExpensesFilter from "./ExpensesFilter";  import Card from "../UI/Card";  import "./Expenses.css";  const Expenses = (props) => {  const [filteredYear, setFilteredYear] = useState("2020");  const filterChangeHandler = (selectedYear) => {  setFilteredYear(selectedYear);  };  return (  <div>  <Card className="expenses">  <ExpensesFilter  selected={filteredYear}  onChangeFilter={filterChangeHandler}  />  {props.items.map((expense) => (  <ExpenseItem  **key={expense.id}**  title={expense.title}  amount={expense.amount}  date={expense.date}  />  ))}  </Card>  </div>  );  };  export default Expenses; |

If you did not have a unique ID, you could use the second argument which you get here for the function you pass to map(), which is an automatically managed index. Though, it is discouraged to use that since with that you can still run into bugs because the index for a given item is always the same and not directly attached to the content of the item. For the ID that's different, as every item with certain content has a clear unique ID.

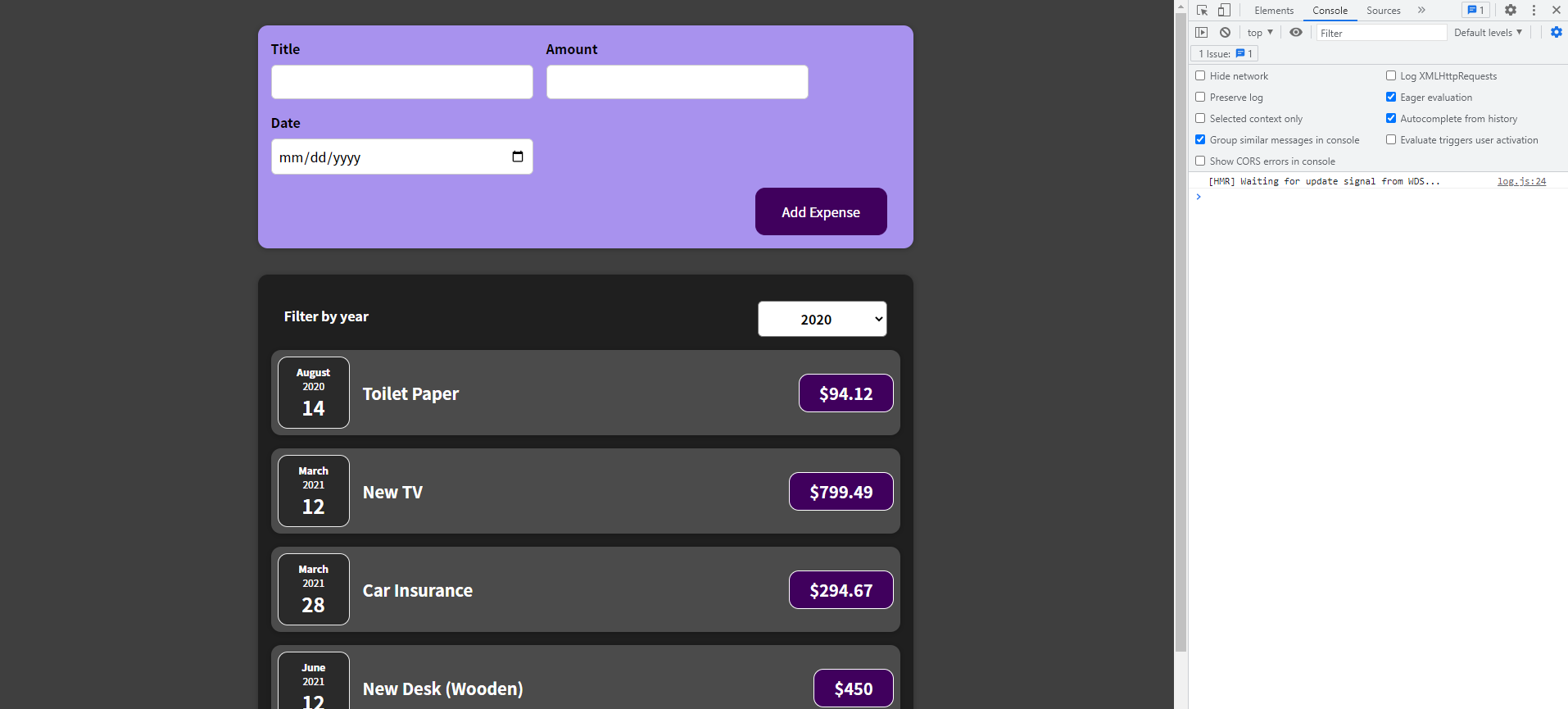
|  |
| --- |
| src/components/Expenses/Expenses.js |
| import React, { useState } from "react";  import ExpenseItem from "./ExpenseItem";  import ExpensesFilter from "./ExpensesFilter";  import Card from "../UI/Card";  import "./Expenses.css";  const Expenses = (props) => {  const [filteredYear, setFilteredYear] = useState("2020");  const filterChangeHandler = (selectedYear) => {  setFilteredYear(selectedYear);  };  return (  <div>  <Card className="expenses">  <ExpensesFilter  selected={filteredYear}  onChangeFilter={filterChangeHandler}  />  {props.items.map((expense, **index**) => (  <ExpenseItem  key={**index**}  title={expense.title}  amount={expense.amount}  date={expense.date}  />  ))}  </Card>  </div>  );  };  export default Expenses; |

In reality, it turns out that in most scenarios you do have some unique ID because typically you are rendering some data which comes from a database or anything like that. And there you work with unique IDs anyway. So finding some unique identifier is no problem, and you can use any primitive value as a unique identifier.

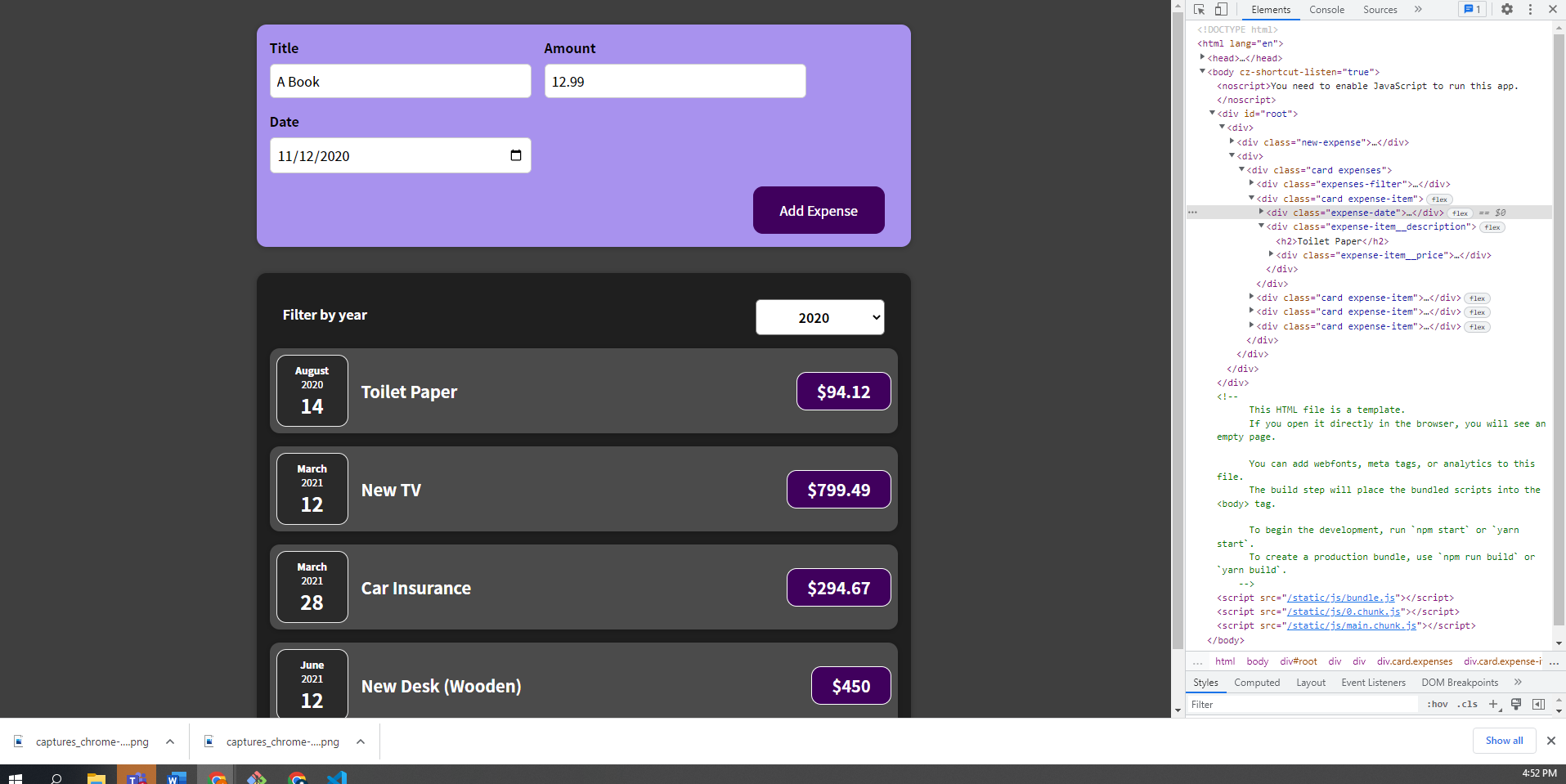
Now, with the key prop pointing at expense.id,

|  |
| --- |
| src/components/Expenses/Expenses.js |
| import React, { useState } from "react";  import ExpenseItem from "./ExpenseItem";  import ExpensesFilter from "./ExpensesFilter";  import Card from "../UI/Card";  import "./Expenses.css";  const Expenses = (props) => {  const [filteredYear, setFilteredYear] = useState("2020");  const filterChangeHandler = (selectedYear) => {  setFilteredYear(selectedYear);  };  return (  <div>  <Card className="expenses">  <ExpensesFilter  selected={filteredYear}  onChangeFilter={filterChangeHandler}  />  {props.items.map((expense) => (  <ExpenseItem  **key={expense.id}**  title={expense.title}  amount={expense.amount}  date={expense.date}  />  ))}  </Card>  </div>  );  };  export default Expenses; |

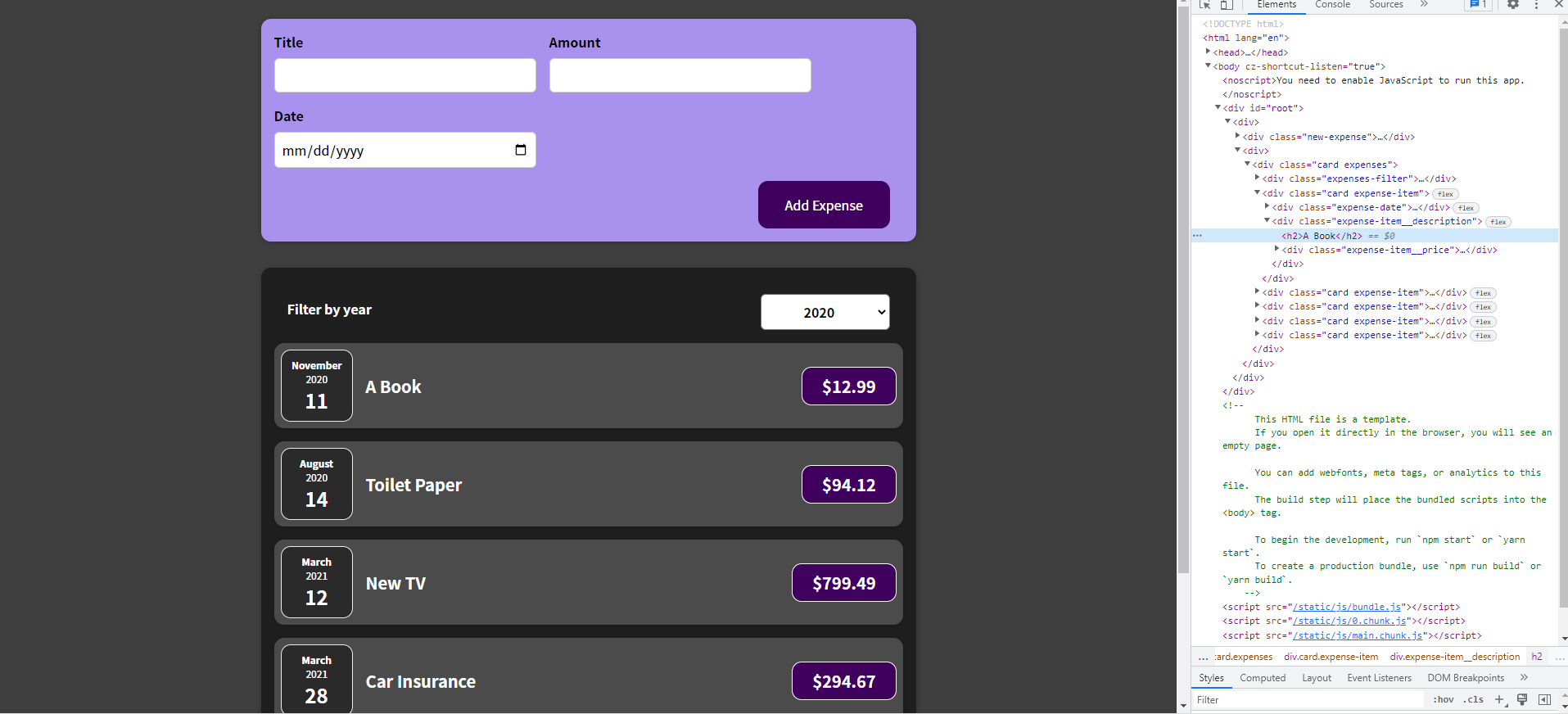
if we reload the page, we no longer get that warning.



If we inspect the first item in the list and add a new expense,



We see the div with the item just added at the beginning of the list of divs correctly, as it should be, because now React is able to uniquely identify all these items, and it is therefore aware, not just how long the array is, but also which items should be placed. It's also able to update this list in a more efficient manner.



You should always add a key prop when mapping out lists of items.

### Assignment 3: Time to Practice: Working with Lists

I need to make the year filter work, so that if a user selects a year value, the list is filtered correctly. So, if I selected an expense from 2021, I only see expenses showing up from 2021 in the rendered list of expenses.

Two hints while working on this:

1. You can filter arrays with the filter method. Like map, filter is a built-in method for arrays in JavaScript.
2. You should not think to complicated here. You should not change the overall expenses array. Instead, you may want to just derive a new array based on the full expenses array. The new array should be a subset of the expenses array for the chosen filter.

### 67. Outputting Conditional Content

Our filter is working but we can still select values from the dropdown where we don't have an expense for the particular year selected. We might want to show a message in such cases where we select years where we don't have expenses for the particular year selected.

Conditional content is about rendering different output under different conditions. In Expenses.js, we render our list of ExpenseItems, but if our filteredExpenses array turns out to be empty, we render nothing. We might want to render a message telling the user that we have no items for the chosen filter. For that, we want to render content conditionally.

We can add a dynamic expression in our JSX code and simply use a condition. We are not talking about an if condition or for loops are not allowed between curly braces because they are too long, but we can use a ternary expression. We can check if filteredExpenses.length is equal to 0 meaning that we have no items in that filteredExpenses array. If that's the case, after the question mark, which is the default JavaScript ternary expression syntax, we render a simple message in a paragraph where we say "No expenses found." Otherwise, after the colon, we render our list.

|  |
| --- |
| src/components/Expenses/Expenses.js |
| import React, { useState } from "react";  import ExpenseItem from "./ExpenseItem";  import ExpensesFilter from "./ExpensesFilter";  import Card from "../UI/Card";  import "./Expenses.css";  const Expenses = (props) => {  const [filteredYear, setFilteredYear] = useState("2020");  const filterChangeHandler = (selectedYear) => {  setFilteredYear(selectedYear);  };  const filteredExpenses = props.items.filter((expense) => {  return expense.date.getFullYear().toString() === filteredYear;  });  return (  <div>  <Card className="expenses">  <ExpensesFilter  selected={filteredYear}  onChangeFilter={filterChangeHandler}  />  **{filteredExpenses.length === 0 ? (**  **<p>No expenses Found</p>**  **) : (**  **filteredExpenses.map((expense) => (**  **<ExpenseItem**  **key={expense.id}**  **title={expense.title}**  **amount={expense.amount}**  **date={expense.date}**  **/>**  **))**  **)}**  </Card>  </div>  );  };  export default Expenses; |

Long ternary expressions like this can be a bit hard to read, so we can simplify this by doing the following:

{filteredExpenses.length === 0 && <p>No expenses found.</p>

We can add the and operator and then use the JSX content we want to render if the length of filteredExpenses is equal to 0. In JavaScript, if you use the && (and) operator, it will return the part after the && operator as a result of the overall check if the first condition (if the length of filteredExpenses is equal to 0) is met. It moves onto the part after the and operator and it then returns that value.

We can now do this same thing when filteredExpenses is greater than 0.

|  |
| --- |
| src/components/Expenses/Expenses.js |
| import React, { useState } from "react";  import ExpenseItem from "./ExpenseItem";  import ExpensesFilter from "./ExpensesFilter";  import Card from "../UI/Card";  import "./Expenses.css";  const Expenses = (props) => {  const [filteredYear, setFilteredYear] = useState("2020");  const filterChangeHandler = (selectedYear) => {  setFilteredYear(selectedYear);  };  const filteredExpenses = props.items.filter((expense) => {  return expense.date.getFullYear().toString() === filteredYear;  });  return (  <div>  <Card className="expenses">  <ExpensesFilter  selected={filteredYear}  onChangeFilter={filterChangeHandler}  />  {filteredExpenses.length === 0 && <p>No expenses found.</p>}  **{filteredExpenses.length > 0 &&**  **filteredExpenses.map((expense) => (**  **<ExpenseItem**  **key={expense.id}**  **title={expense.title}**  **amount={expense.amount}**  **date={expense.date}**  **/>**  **))}**  </Card>  </div>  );  };  export default Expenses; |

This might even be too much logic in the JSX code, so we have an alternative way of handling this. We can create a variable with a default value like so:

|  |
| --- |
| src/components/Expenses/Expenses.js |
| import React, { useState } from "react";  import ExpenseItem from "./ExpenseItem";  import ExpensesFilter from "./ExpensesFilter";  import Card from "../UI/Card";  import "./Expenses.css";  const Expenses = (props) => {  const [filteredYear, setFilteredYear] = useState("2020");  const filterChangeHandler = (selectedYear) => {  setFilteredYear(selectedYear);  };  const filteredExpenses = props.items.filter((expense) => {  return expense.date.getFullYear().toString() === filteredYear;  });  **let expensesContent = <p>No expenses found.</p>**  return (  <div>  <Card className="expenses">  <ExpensesFilter  selected={filteredYear}  onChangeFilter={filterChangeHandler}  />  {filteredExpenses.length === 0 && <p>No expenses found.</p>}  {filteredExpenses.length > 0 &&  filteredExpenses.map((expense) => (  <ExpenseItem  key={expense.id}  title={expense.title}  amount={expense.amount}  date={expense.date}  />  ))}  </Card>  </div>  );  };  export default Expenses; |

We can store JSX content in variables. We store it in a variable here specifically because now we can add an if check before we return. We can check if filteredExpenses.length is greater than 0. If filteredExpenses is greater than 0, we can set expensesContent to the value of the map call.

|  |
| --- |
| src/components/Expenses/Expenses.js |
| import React, { useState } from "react";  import ExpenseItem from "./ExpenseItem";  import ExpensesFilter from "./ExpensesFilter";  import Card from "../UI/Card";  import "./Expenses.css";  const Expenses = (props) => {  const [filteredYear, setFilteredYear] = useState("2020");  const filterChangeHandler = (selectedYear) => {  setFilteredYear(selectedYear);  };  const filteredExpenses = props.items.filter((expense) => {  return expense.date.getFullYear().toString() === filteredYear;  });  let expensesContent = <p>No expenses found.</p>  **if (filteredExpenses.length > 0){**  **expensesContent = filteredExpenses.map((expense) => (**  **<ExpenseItem**  **key={expense.id}**  **title={expense.title}**  **amount={expense.amount}**  **date={expense.date}**  **/>**  **))**  **}**  return (  <div>  <Card className="expenses">  <ExpensesFilter  selected={filteredYear}  onChangeFilter={filterChangeHandler}  />  {filteredExpenses.length === 0 && <p>No expenses found.</p>}  {filteredExpenses.length > 0 &&  filteredExpenses.map((expense) => (  <ExpenseItem  key={expense.id}  title={expense.title}  amount={expense.amount}  date={expense.date}  />  ))}  </Card>  </div>  );  };  export default Expenses; |

In our returned JSX code, we can get rid of all that logic, and we simply add a dynamic expression where we point at expensesContent.

|  |
| --- |
| src/components/Expenses/Expenses.js |
| import React, { useState } from "react";  import ExpenseItem from "./ExpenseItem";  import ExpensesFilter from "./ExpensesFilter";  import Card from "../UI/Card";  import "./Expenses.css";  const Expenses = (props) => {  const [filteredYear, setFilteredYear] = useState("2020");  const filterChangeHandler = (selectedYear) => {  setFilteredYear(selectedYear);  };  const filteredExpenses = props.items.filter((expense) => {  return expense.date.getFullYear().toString() === filteredYear;  });  let expensesContent = <p>No expenses found.</p>  if (filteredExpenses.length > 0){  expensesContent = filteredExpenses.map((expense) => (  <ExpenseItem  key={expense.id}  title={expense.title}  amount={expense.amount}  date={expense.date}  />  ))  }  return (  <div>  <Card className="expenses">  <ExpensesFilter  selected={filteredYear}  onChangeFilter={filterChangeHandler}  />  **{expensesContent}**  </Card>  </div>  );  };  export default Expenses; |

### 68. Adding Conditional Return Statements

We are going to extract the list logic into a new component to make the Expenses component a little leaner. We will create an "ExpensesList.js" file and an "ExpensesList.css" file. We will copy the css code given into "ExpenseList.css" and then in "ExpensesList.js" we will write the following code:

|  |
| --- |
| src/components/Expenses/ExpensesList.js |
| import React from 'react';  const ExpensesList = props => {  }  export default ExpensesList; |

I will take the fallback content and the if check and move that into the ExpensesList component:

|  |
| --- |
| src/components/Expenses/ExpensesList.js |
| import React from 'react';  import ExpenseItem from './ExpenseItem';  const ExpensesList = props => {  let expensesContent = <p>No expenses found.</p>  if (filteredExpenses.length > 0){  expensesContent = filteredExpenses.map((expense) => (  <ExpenseItem  key={expense.id}  title={expense.title}  amount={expense.amount}  date={expense.date}  />  ))  }  }  export default ExpensesList; |

The variable filteredExpenses no longer exists here. We just use props.items instead.

|  |
| --- |
| src/components/Expenses/ExpensesList.js |
| import React from 'react';  import ExpenseItem from './ExpenseItem';  const ExpensesList = props => {  let expensesContent = <p>No expenses found.</p>  if (**props.items**.length > 0){  expensesContent = **props.items**.map((expense) => (  <ExpenseItem  key={expense.id}  title={expense.title}  amount={expense.amount}  date={expense.date}  />  ))  }  }  export default ExpensesList; |

This means that in Expenses.js we now have the following:

|  |
| --- |
| src/components/Expenses/Expenses.js |
| import React, { useState } from "react";  import ExpensesFilter from "./ExpensesFilter";  import ExpensesList from "./ExpensesList";  import Card from "../UI/Card";  import "./Expenses.css";  const Expenses = (props) => {  const [filteredYear, setFilteredYear] = useState("2020");  const filterChangeHandler = (selectedYear) => {  setFilteredYear(selectedYear);  };  const filteredExpenses = props.items.filter((expense) => {  return expense.date.getFullYear().toString() === filteredYear;  });  return (  <div>  <Card className="expenses">  <ExpensesFilter  selected={filteredYear}  onChangeFilter={filterChangeHandler}  />  **<ExpensesList items={filteredExpenses}/>**  </Card>  </div>  );  };  export default Expenses; |

In "ExpensesList.js" we want to make some changes. We want to return some JSX, and we will return an unordered list, giving it a className of "expenses-list".

|  |
| --- |
| src/components/Expenses/ExpensesList.js |
| import React from 'react';  import ExpenseItem from './ExpenseItem';  import './ExpensesList.css';  const ExpensesList = props => {  let expensesContent = <p>No expenses found.</p>  if (props.items.length > 0){  expensesContent = props.items.map((expense) => (  <ExpenseItem  key={expense.id}  title={expense.title}  amount={expense.amount}  date={expense.date}  />  ))  }  **return <ul className="expensesList">**  **</ul>**  }  export default ExpensesList; |

Inside of the unordered list, I want to output my map here, my ExpenseItem array.

|  |
| --- |
| src/components/Expenses/ExpensesList.js |
| import React from 'react';  import ExpenseItem from './ExpenseItem';  import './ExpensesList.css';  const ExpensesList = props => {  let expensesContent = <p>No expenses found.</p>  if (props.items.length > 0){  expensesContent = props.items.map((expense) => (  <ExpenseItem  key={expense.id}  title={expense.title}  amount={expense.amount}  date={expense.date}  />  ))  }  return <ul className="expensesList">  **{**  **props.items.map((expense) => (**  **<ExpenseItem**  **key={expense.id}**  **title={expense.title}**  **amount={expense.amount}**  **date={expense.date}**  **/>**  **))**  **}**  </ul>  }  export default ExpensesList; |

We will now handle the if check a little differently. We will add the if check, but we will check if props.items.length equals 0, so if we have no items, and if that is the case we will return a totally different JSX snippet because that's yet another way of handling conditional content. If what your component returns changes entirely based on different conditions, you can use this approach. It would not have been appropriate in "Expenses.js" because only a part of the JSX snippet which we returned changed. If your entire JSX content changes when data is missing, you can also add an if check where you return a different JSX block if some condition is met. So here, we can return an h2 element with a className of "expenses-list\_\_fallback" with a text of "Found no expenses." We can also get rid of the variable called expensesContent.

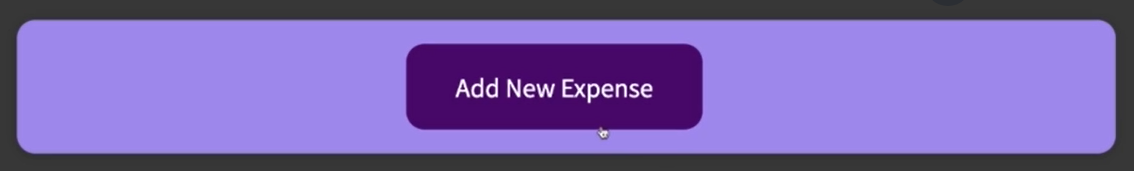
|  |
| --- |
| src/components/Expenses/ExpensesList.js |
| import React from 'react';  import ExpenseItem from './ExpenseItem';  import './ExpensesList.css';  const ExpensesList = props => {  **if (props.items.length === 0) {**  **return <h2 className="expenses-list\_\_fallback">Found no expenses.</h2>**  **}**  return <ul className="expensesList">  {  props.items.map((expense) => (  <ExpenseItem  key={expense.id}  title={expense.title}  amount={expense.amount}  date={expense.date}  />  ))  }  </ul>  }  export default ExpensesList; |

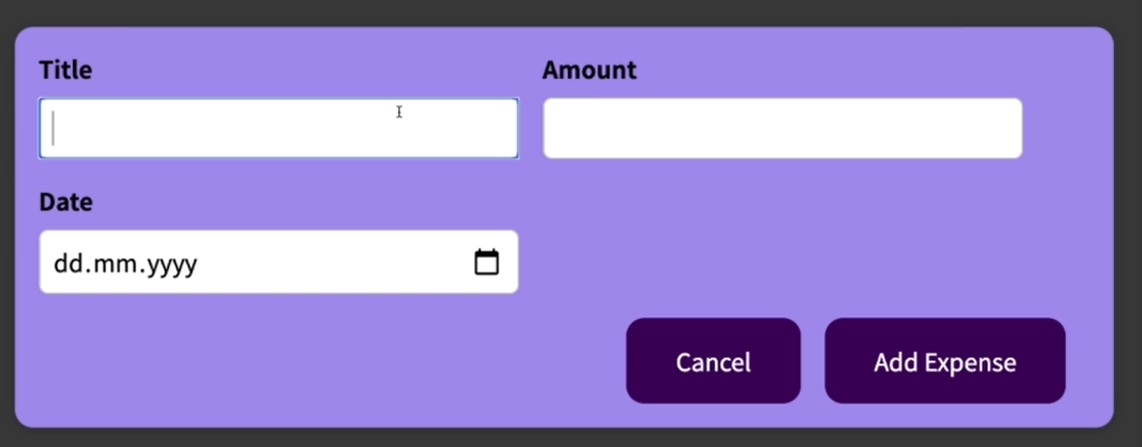
Since we are using an unordered list in the ExpenseList component, for semantic reasons, in the ExpenseItem component, we will switch away from a div being rendered to a list item instead.

|  |
| --- |
| src/components/Expenses/ExpenseItem.js |
| import React, { useState } from "react";  import ExpenseDate from "./ExpenseDate";  import Card from "../UI/Card";  import "./ExpenseItem.css";  const ExpenseItem = (props) => {  return (  **<li>**  <Card className="expense-item">  <ExpenseDate date={props.date} />  <div className="expense-item\_\_description">  <h2>{props.title}</h2>  <div className="expense-item\_\_price">${props.amount}</div>  </div>  </Card>  **</li>**  );  };  export default ExpenseItem; |

### Assignment 4: Time to Practice: Conditional Content

The form is always currently shown. We need to change the logic to make sure that we have a button that can be shown instead of the form where the button says something like "Add New Expense". When the button is clicked, the button disappears and the form is shown instead, and when the form's add expense button is clicked, the form disappears again, and the button which opens the form is shown again.



You should also add a cancel button to this form, which also makes the form disappear without submitting it. 

#### Plan for Assignment 4: Time to Practice: Conditional Content

In the ExpenseForm component, create a state hook:

const [formVisable, setFormVisable] = useState(false);

Wrap JSX in the return with a JSX fragment.

Check if {if !formVisable{ <button onClick={showForm}>Add Expense</button>}}

In show showForm, call setFormVisable(true);

Create else condition to show the form.

Create a cancel button inside the form, which has an onClick event that points at the function hideForm. In the hideForm function, call setFormVisable(false);

In the submitHandler method, after event.preventDefault();, call setFormVisable(false);

Code I ended up writing:

|  |
| --- |
| src/components/NewExpense/ExpenseForm.js |
| import React, { useState } from "react";  import "./ExpenseForm.css";  const ExpenseForm = (props) => {  const [enteredTitle, setEnteredTitle] = useState("");  const [enteredAmount, setEnteredAmount] = useState("");  const [enteredDate, setEnteredDate] = useState("");  **const [formHidden, setFormHidden] = useState(true);**  const titleChangeHandler = (event) => {  setEnteredTitle(event.target.value);  };  const amountChangeHandler = (event) => {  const dateParts = event.target.value.split("-");  setEnteredAmount(dateParts[0], dateParts[1] + 1, dateParts[2]);  };  const dateChangeHandler = (event) => {  setEnteredDate(event.target.value);  };  // Handle date issue  const dateString = enteredDate;  const dateParts = dateString.split("-");  const aDate = new Date(dateParts[0], dateParts[1] - 1, dateParts[2]);  const submitHandler = (event) => {  event.preventDefault();  **setFormHidden(true);**  const expenseData = {  title: enteredTitle,  amount: enteredAmount,  date: aDate,  };  props.onSaveExpenseData(expenseData);  setEnteredTitle("");  setEnteredAmount("");  setEnteredDate("");  };  **const showForm = () => {**  **setFormHidden(false);**  **}**  **const hideForm = () => {**  **setFormHidden(true);**  **}**    **const expenseButton = <button onClick={showForm}>Add Expense</button>**  **const expenseForm =** <form onSubmit={submitHandler}>  <div className="new-expense\_\_controls">  <div className="new-expense\_\_control">  <label>Title</label>  <input  type="text"  value={enteredTitle}  onChange={titleChangeHandler}  />  </div>  <div className="new-expense\_\_control">  <label>Amount</label>  <input  type="number"  value={enteredAmount}  min="0.01"  step="0.01"  onChange={amountChangeHandler}  />  </div>  <div className="new-expense\_\_control">  <label>Date</label>  <input  type="date"  value={enteredDate}  min="2019-01-01"  max="2022-12-31"  onChange={dateChangeHandler}  />  </div>  </div>  <div className="new-expense\_\_actions">  **<button onClick={hideForm}>Cancel</button>**  <button type="submit">Add Expense</button>  </div>  </form>  return (  **<>**  **{ formHidden ? expenseButton : expenseForm }**  **</>**  );  };  export default ExpenseForm; |

#### Notes from Max's solution video

We have this NewExpense component, and that is where we render ExpenseForm.

|  |
| --- |
| src/components/NewExpense/NewExpense.js |
| import React from 'react';  import ExpenseForm from './ExpenseForm';  import './NewExpense.css';  const NewExpense = (props) => {  const saveExpenseDataHandler = (enteredExpenseData) => {  const expenseData = {  ...enteredExpenseData,  id: Math.random().toString()  };  props.onAddExpense(expenseData);  };  return (  <div className='new-expense'>  **<ExpenseForm onSaveExpenseData={saveExpenseDataHandler} />**  </div>  );  };  export default NewExpense; |

We want to be able to render ExpenseForm conditionally. We want to make sure that it is only shown sometimes, and that in some cases, instead of the ExpenseForm, we are showing a button. So, therefore, in NewExpense, we want to add <button>Add New Expense</button>.

|  |
| --- |
| src/components/NewExpense/NewExpense.js |
| import React from "react";  import ExpenseForm from "./ExpenseForm";  import "./NewExpense.css";  const NewExpense = (props) => {  const saveExpenseDataHandler = (enteredExpenseData) => {  const expenseData = {  ...enteredExpenseData,  id: Math.random().toString(),  };  props.onAddExpense(expenseData);  };  return (  <div className="new-expense">  **<button>Add New Expense</button>**  <ExpenseForm onSaveExpenseData={saveExpenseDataHandler} />  </div>  );  };  export default NewExpense; |

When the "Add New Expense" button is clicked, we want to show the form, and when the form is submitted or canceled, the button should be shown. That means that we need to register a new state here because we'll have a brand-new state for our application or this part of the application.

We have the state that we're currently adding or editing in expense. We have an editing state where the form should be open and we have another state where we are currently not working on an expense where the button should be shown instead. We'll have a brand-new state which should be conditionally rendered on the screen, and, therefore, we need to add state to our component, a new state variable. Let's import useState from 'react';

|  |
| --- |
| src/components/NewExpense/NewExpense.js |
| import React, **{useState}** from "react";  import ExpenseForm from "./ExpenseForm";  import "./NewExpense.css";  const NewExpense = (props) => {  const saveExpenseDataHandler = (enteredExpenseData) => {  const expenseData = {  ...enteredExpenseData,  id: Math.random().toString(),  };  props.onAddExpense(expenseData);  };  return (  <div className="new-expense">  <button>Add New Expense</button>  <ExpenseForm onSaveExpenseData={saveExpenseDataHandler} />  </div>  );  };  export default NewExpense; |

We will call the state, and we basically just need a true or false state which states whether the form shall be shown or not. Initially, we could start with false, and we could name this state isEditing and setIsEditing, which is the state updating function.

|  |
| --- |
| src/components/NewExpense/NewExpense.js |
| import React, {useState} from "react";  import ExpenseForm from "./ExpenseForm";  import "./NewExpense.css";  const NewExpense = (props) => {  **const [isEditing, setIsEditing] = useState(false);**  const saveExpenseDataHandler = (enteredExpenseData) => {  const expenseData = {  ...enteredExpenseData,  id: Math.random().toString(),  };  props.onAddExpense(expenseData);  };  return (  <div className="new-expense">  <button>Add New Expense</button>  <ExpenseForm onSaveExpenseData={saveExpenseDataHandler} />  </div>  );  };  export default NewExpense; |

We then add a new function called startEditingHandler, which calls setIsEditing(true);, which sets isEditing to true.

|  |
| --- |
| src/components/NewExpense/NewExpense.js |
| import React, {useState} from "react";  import ExpenseForm from "./ExpenseForm";  import "./NewExpense.css";  const NewExpense = (props) => {  const [isEditing, setIsEditing] = useState(false);  const saveExpenseDataHandler = (enteredExpenseData) => {  const expenseData = {  ...enteredExpenseData,  id: Math.random().toString(),  };  props.onAddExpense(expenseData);  };  **const startEditingHandler = () => {**  **setIsEditing(true);**  **}**  return (  <div className="new-expense">  <button>Add New Expense</button>  <ExpenseForm onSaveExpenseData={saveExpenseDataHandler} />  </div>  );  };  export default NewExpense; |

The function startEditingHandler should be triggered when the "Add New Expense" button is clicked, which then in turn should lead to the ExpenseForm showing up. Therefore, on this button, we can add the onClick prop and point at the startEditingHandler function when it's clicked, so that isEditing is set to true when this button is clicked.

|  |
| --- |
| src/components/NewExpense/NewExpense.js |
| import React, {useState} from "react";  import ExpenseForm from "./ExpenseForm";  import "./NewExpense.css";  const NewExpense = (props) => {  const [isEditing, setIsEditing] = useState(false);  const saveExpenseDataHandler = (enteredExpenseData) => {  const expenseData = {  ...enteredExpenseData,  id: Math.random().toString(),  };  props.onAddExpense(expenseData);  };  const startEditingHandler = () => {  setIsEditing(true);  }  return (  <div className="new-expense">  <button **onClick={startEditingHandler}**>Add New Expense</button>  <ExpenseForm onSaveExpenseData={saveExpenseDataHandler} />  </div>  );  };  export default NewExpense; |

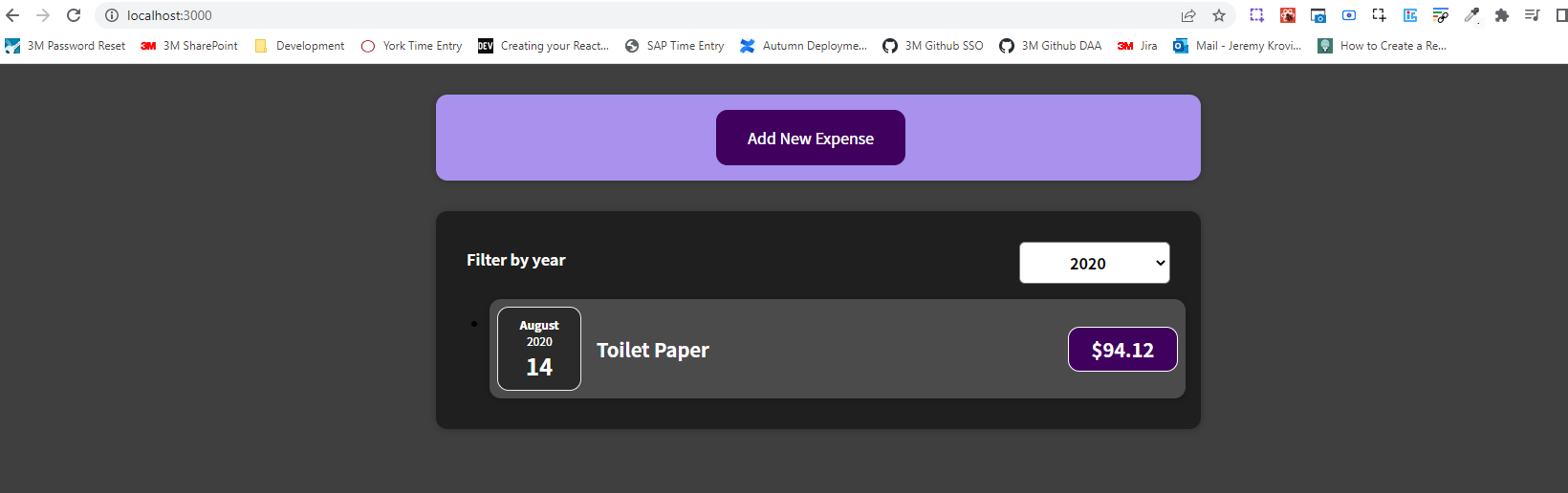
Now, we can use the isEditing state to control which of the two elements here—the button or the ExpenseForm—is shown. We want to show the button if we are not editing, hence the exclamation mark at the beginning, and we will do this in one line using &&.

|  |
| --- |
| src/components/NewExpense/NewExpense.js |
| import React, {useState} from "react";  import ExpenseForm from "./ExpenseForm";  import "./NewExpense.css";  const NewExpense = (props) => {  const [isEditing, setIsEditing] = useState(false);  const saveExpenseDataHandler = (enteredExpenseData) => {  const expenseData = {  ...enteredExpenseData,  id: Math.random().toString(),  };  props.onAddExpense(expenseData);  };  const startEditingHandler = () => {  setIsEditing(true);  }  return (  <div className="new-expense">  **{!isEditing &&** <button onClick={startEditingHandler}>Add New Expense</button>**}**  <ExpenseForm onSaveExpenseData={saveExpenseDataHandler} />  </div>  );  };  export default NewExpense; |

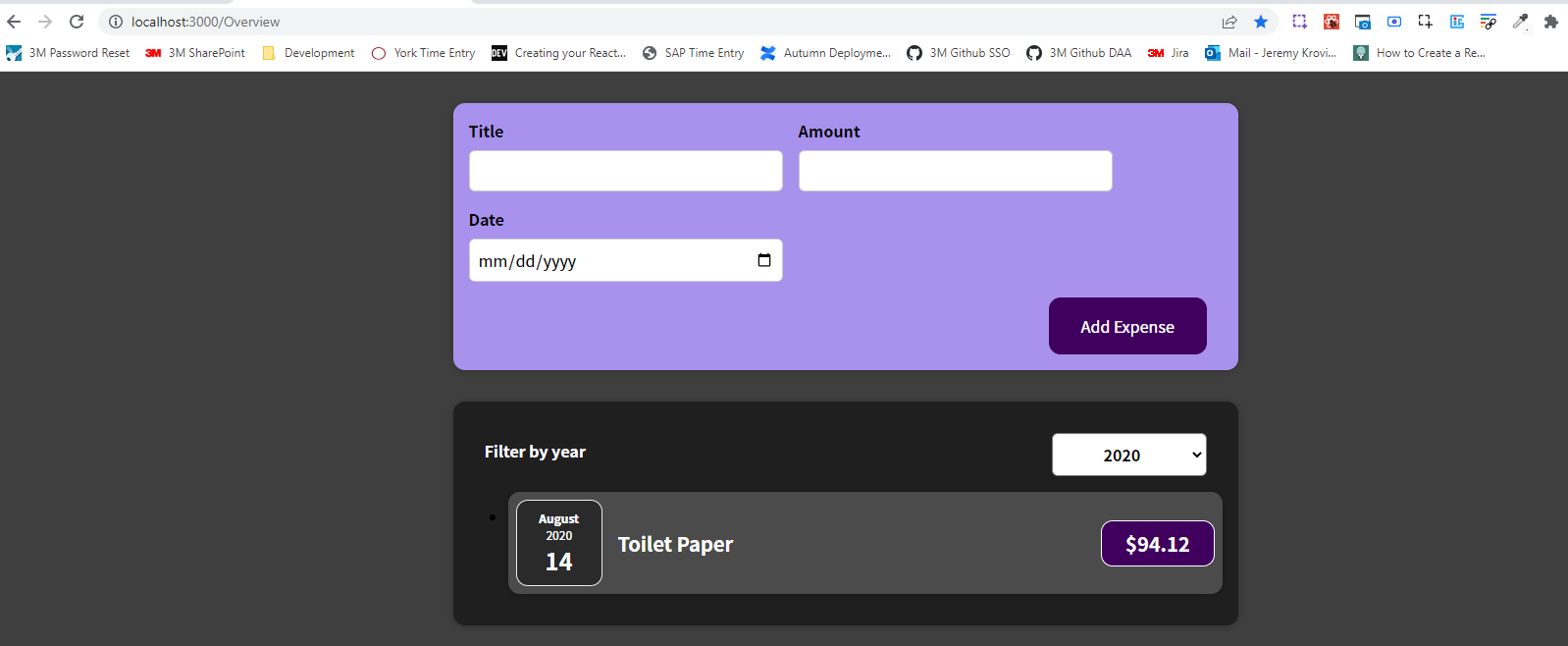
Of course you could also work with a variable that has a default value, or an if check, or with a ternary expression. Instead, we will have two dynamic expressions. Check if isEditing is false, hence we show the button, and check if isEditing is true, in which case we show the ExpenseForm component.

|  |
| --- |
| src/components/NewExpense/NewExpense.js |
| import React, {useState} from "react";  import ExpenseForm from "./ExpenseForm";  import "./NewExpense.css";  const NewExpense = (props) => {  const [isEditing, setIsEditing] = useState(false);  const saveExpenseDataHandler = (enteredExpenseData) => {  const expenseData = {  ...enteredExpenseData,  id: Math.random().toString(),  };  props.onAddExpense(expenseData);  };  const startEditingHandler = () => {  setIsEditing(true);  }  return (  <div className="new-expense">  {!isEditing && <button onClick={startEditingHandler}>Add New Expense</button>}  **{isEditing &&** <ExpenseForm onSaveExpenseData={saveExpenseDataHandler} />**}**  </div>  );  };  export default NewExpense; |

Now if we save and reload we will see just the button



If we click the button, we will see the form



Inside of the form, we now also want to add a "Cancel" button and make sure that when the form is submitted, we also stop editing. In ExpenseForm, we can add a button side-by-side with the "Add Expense" button and give it the text "Cancel", set type="button", so that the button does not submit the form when it is clicked and add a onClick handler to execute a function when the "Cancel" button is pressed.

|  |
| --- |
| src/components/NewExpense/ExpenseForm.js |
| import React, { useState } from "react";  import "./ExpenseForm.css";  const ExpenseForm = (props) => {  const [enteredTitle, setEnteredTitle] = useState("");  const [enteredAmount, setEnteredAmount] = useState("");  const [enteredDate, setEnteredDate] = useState("");  // const [userInput, setUserInput] = useState({  // enteredTitle: '',  // enteredAmount: '',  // enteredDate: ''  // });  const titleChangeHandler = (event) => {  setEnteredTitle(event.target.value);  // setUserInput({  // ...userInput,  // enteredTitle: event.target.value,  // })  // setUserInput((prevState) => {  // return { ...prevState, enteredTitle: event.target.value}  // })  };  const amountChangeHandler = (event) => {  const dateParts = event.target.value.split("-");  setEnteredAmount(dateParts[0], dateParts[1] + 1, dateParts[2]);  // setUserInput({  // ...userInput,  // enteredAmount: event.target.value,  // })  };  const dateChangeHandler = (event) => {  setEnteredDate(event.target.value);  // setUserInput({  // ...userInput,  // enteredDate: event.target.value,  // })  };  // Handle date issue  const dateString = enteredDate;  const dateParts = dateString.split("-");  const aDate = new Date(dateParts[0], dateParts[1] - 1, dateParts[2]);  const submitHandler = (event) => {  event.preventDefault();  const expenseData = {  title: enteredTitle,  amount: enteredAmount,  date: aDate,  };  props.onSaveExpenseData(expenseData);  setEnteredTitle("");  setEnteredAmount("");  setEnteredDate("");  };  return (  <form onSubmit={submitHandler}>  <div className="new-expense\_\_controls">  <div className="new-expense\_\_control">  <label>Title</label>  <input  type="text"  value={enteredTitle}  onChange={titleChangeHandler}  />  </div>  <div className="new-expense\_\_control">  <label>Amount</label>  <input  type="number"  value={enteredAmount}  min="0.01"  step="0.01"  onChange={amountChangeHandler}  />  </div>  <div className="new-expense\_\_control">  <label>Date</label>  <input  type="date"  value={enteredDate}  min="2019-01-01"  max="2022-12-31"  onChange={dateChangeHandler}  />  </div>  </div>  <div className="new-expense\_\_actions">  **<button type="button" onClick={}>Cancel</button>**  <button type="submit">Add Expense</button>  </div>  </form>  );  };  export default ExpenseForm; |

We don't want to execute a function in the ExpenseForm component but in NewExpense instead. We will define the function stopEditingHandler where we call setIsEditing(false), which sets isEditing to false.

|  |
| --- |
| src/components/NewExpense/NewExpense.js |
| import React, {useState} from "react";  import ExpenseForm from "./ExpenseForm";  import "./NewExpense.css";  const NewExpense = (props) => {  const [isEditing, setIsEditing] = useState(false);  const saveExpenseDataHandler = (enteredExpenseData) => {  const expenseData = {  ...enteredExpenseData,  id: Math.random().toString(),  };  props.onAddExpense(expenseData);  };  const startEditingHandler = () => {  setIsEditing(true);  }  **const stopEditingHandler = () => {**  **setIsEditing(false);**  **}**  return (  <div className="new-expense">  {!isEditing && <button onClick={startEditingHandler}>Add New Expense</button>}  {isEditing && <ExpenseForm onSaveExpenseData={saveExpenseDataHandler} />}  </div>  );  };  export default NewExpense; |

The stopEditingHandler function is the function which we in the end want to trigger if the "Cancel" button is pressed in the ExpenseForm component. We just need to pass a pointer at stopEditingHandler down to the ExpenseForm component. In the call to the ExpenseForm component within the NewExpense component, we add a new prop called onCancel and point at the stopEditingHandler function, so that the function stopEditingHandler function is passed as a value to the onCancel prop in the ExpenseForm component.

|  |
| --- |
| src/components/NewExpense/NewExpense.js |
| import React, {useState} from "react";  import ExpenseForm from "./ExpenseForm";  import "./NewExpense.css";  const NewExpense = (props) => {  const [isEditing, setIsEditing] = useState(false);  const saveExpenseDataHandler = (enteredExpenseData) => {  const expenseData = {  ...enteredExpenseData,  id: Math.random().toString(),  };  props.onAddExpense(expenseData);  };  const startEditingHandler = () => {  setIsEditing(true);  }  const stopEditingHandler = () => {  setIsEditing(false);  }  return (  <div className="new-expense">  {!isEditing && <button onClick={startEditingHandler}>Add New Expense</button>}  {isEditing &&  <ExpenseForm  onSaveExpenseData={saveExpenseDataHandler}  **onCancel={stopEditingHandler}** />}  </div>  );  };  export default NewExpense; |

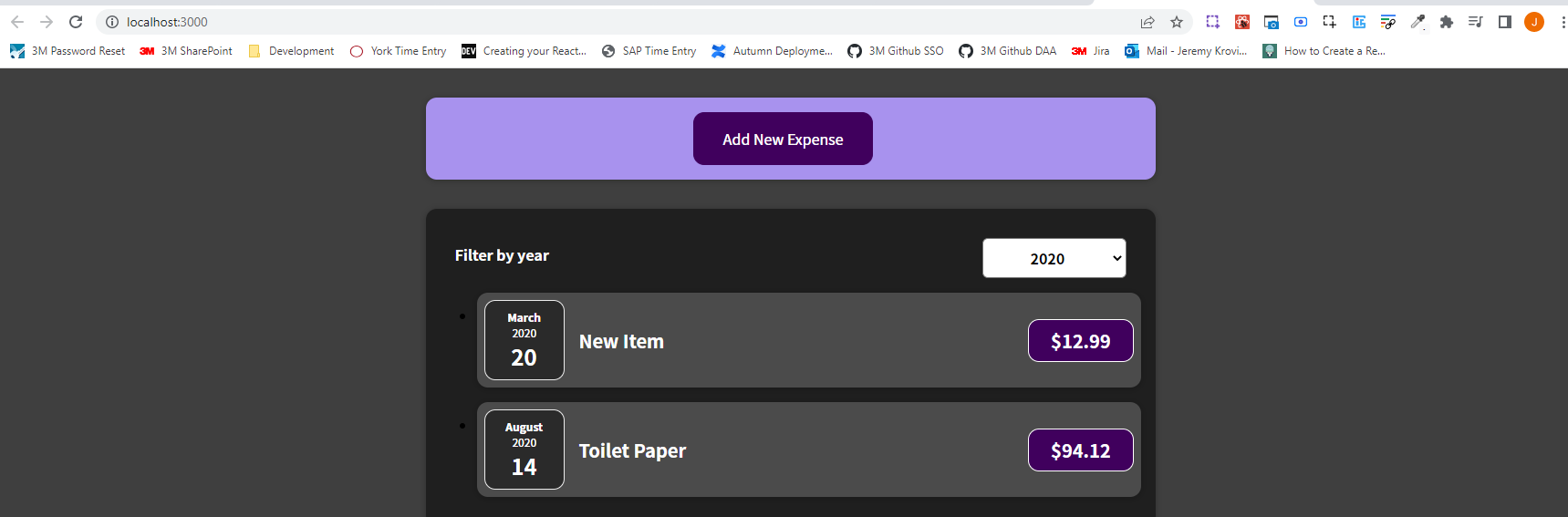
Then, in the ExpenseForm component, we can now simply point at props.onCancel for a click on the "Cancel" button. So that the function onCancel, which we receive on the onCancel prop is forwarded to the onClick prop of this button, so that for a click on the button, the function stored in onCancel is executed, which of course is the stopEditingHandler function.

|  |
| --- |
| src/components/NewExpense/ExpenseForm.js |
| import React, { useState } from "react";  import "./ExpenseForm.css";  const ExpenseForm = (props) => {  const [enteredTitle, setEnteredTitle] = useState("");  const [enteredAmount, setEnteredAmount] = useState("");  const [enteredDate, setEnteredDate] = useState("");  // const [userInput, setUserInput] = useState({  // enteredTitle: '',  // enteredAmount: '',  // enteredDate: ''  // });  const titleChangeHandler = (event) => {  setEnteredTitle(event.target.value);  // setUserInput({  // ...userInput,  // enteredTitle: event.target.value,  // })  // setUserInput((prevState) => {  // return { ...prevState, enteredTitle: event.target.value}  // })  };  const amountChangeHandler = (event) => {  const dateParts = event.target.value.split("-");  setEnteredAmount(dateParts[0], dateParts[1] + 1, dateParts[2]);  // setUserInput({  // ...userInput,  // enteredAmount: event.target.value,  // })  };  const dateChangeHandler = (event) => {  setEnteredDate(event.target.value);  // setUserInput({  // ...userInput,  // enteredDate: event.target.value,  // })  };  // Handle date issue  const dateString = enteredDate;  const dateParts = dateString.split("-");  const aDate = new Date(dateParts[0], dateParts[1] - 1, dateParts[2]);  const submitHandler = (event) => {  event.preventDefault();  const expenseData = {  title: enteredTitle,  amount: enteredAmount,  date: aDate,  };  props.onSaveExpenseData(expenseData);  setEnteredTitle("");  setEnteredAmount("");  setEnteredDate("");  };  return (  <form onSubmit={submitHandler}>  <div className="new-expense\_\_controls">  <div className="new-expense\_\_control">  <label>Title</label>  <input  type="text"  value={enteredTitle}  onChange={titleChangeHandler}  />  </div>  <div className="new-expense\_\_control">  <label>Amount</label>  <input  type="number"  value={enteredAmount}  min="0.01"  step="0.01"  onChange={amountChangeHandler}  />  </div>  <div className="new-expense\_\_control">  <label>Date</label>  <input  type="date"  value={enteredDate}  min="2019-01-01"  max="2022-12-31"  onChange={dateChangeHandler}  />  </div>  </div>  <div className="new-expense\_\_actions">  <button type="button" **onClick={props.onCancel}**>Cancel</button>  <button type="submit">Add Expense</button>  </div>  </form>  );  };  export default ExpenseForm; |

We also want to close the form if it's submitted, so in saveExpenseDataHandler within the NewExpense component, we call setIsEditing(false);, which sets isEditing to false.

|  |
| --- |
| src/components/NewExpense/NewExpense.js |
| import React, {useState} from "react";  import ExpenseForm from "./ExpenseForm";  import "./NewExpense.css";  const NewExpense = (props) => {  const [isEditing, setIsEditing] = useState(false);  const saveExpenseDataHandler = (enteredExpenseData) => {  const expenseData = {  ...enteredExpenseData,  id: Math.random().toString(),  };  props.onAddExpense(expenseData);  **setIsEditing(false);**  };  const startEditingHandler = () => {  setIsEditing(true);  }  const stopEditingHandler = () => {  setIsEditing(false);  }  return (  <div className="new-expense">  {!isEditing && <button onClick={startEditingHandler}>Add New Expense</button>}  {isEditing &&  <ExpenseForm  onSaveExpenseData={saveExpenseDataHandler}  onCancel={stopEditingHandler} />}  </div>  );  };  export default NewExpense; |

We our now able to reload the page and have the form close when a new item is submitted. The form also closes when the "Cancel" button is clicked.



### 69. Demo App: Adding a Chart

One key feature of our application is missing. The feature that is missing is the chart where we can see a graphical representation of our expenses per month. We will add a new components folder, which will be called Chart because it will hold all the components that are related to the chart. The chart will be split into two main components. One component is the chart itself, which will be in a file called "Chart.js" and another component for the bars inside of the chart which will be in a file called "ChartBar.js". We'll also have styles, so we'll add "Chart.css" and "ChartBar.css". The CSS styling is given to us.

We will start with the "Chart.js" file, with the Chart itself. We will do the basic setup for the Chart component.

|  |
| --- |
| src/components/Chart/Chart.js |
| import React from 'react';  const Chart = props => {};  export default Chart; |

In the Chart we want to render all these ChartBars. How are we going to do that? How are we going to structure the JSX code inside of this component and how should all of that work? We're going to import the ChartBar component, which now is empty.

|  |
| --- |
| src/components/Chart/Chart.js |
| import React from 'react';  **import ChartBar from './ChartBar';**  const Chart = props => {};  export default Chart; |

We then want to return some JSX code where we have all these ChartBars.

|  |
| --- |
| src/components/Chart/Chart.js |
| import React from 'react';  import ChartBar from './ChartBar';  const Chart = props => {  **return**  };  export default Chart; |

We can setup a div with a className of "chart" to apply that styling.

|  |
| --- |
| src/components/Chart/Chart.js |
| import React from 'react';  import ChartBar from './ChartBar';  const Chart = props => {  return **<div className="chart"></div>**  };  export default Chart; |

We then also import './Chart.css';

|  |
| --- |
| src/components/Chart/Chart.js |
| import React from 'react';  import ChartBar from './ChartBar';  **import './Chart.css';**  const Chart = props => {  return <div className="chart"></div>  };  export default Chart; |

Inside of the div we now want to have the bars. We could simply render 12 individual ChartBars for the 12 months that we have. We will create a more flexible chart, which is not restricted to months and to 12 dataPoints. Instead, we could say that when the Chart component is being used somewhere in our application, we want to receive the dataPoints that should be plotted as props, so that the Chart component is configurable and the components that use the Chart component can decide how many dataPoints with which values should be rendered. And, therefore, the ChartBars will be output dynamically by going through an array of dataPoints and mapping every dataPoint to a ChartBar. So, on props, we could expect a dataPoints prop, and of course, that name is up to you since it's your component.

|  |
| --- |
| src/components/Chart/Chart.js |
| import React from 'react';  import ChartBar from './ChartBar';  import './Chart.css';  const Chart = props => {  return <div className="chart">  **{props.dataPoints}**  </div>  };  export default Chart; |

I expect props.dataPoints to hold a value which is an array. Hence, we can call map on props.dataPoints.

|  |
| --- |
| src/components/Chart/Chart.js |
| import React from 'react';  import ChartBar from './ChartBar';  import './Chart.css';  const Chart = props => {  return <div className="chart">  {props.dataPoints**.map()**}  </div>  };  export default Chart; |

We can then map every single dataPoint into a ChartBar component, so that we create as many ChartBar components as we have dataPoints.

|  |
| --- |
| src/components/Chart/Chart.js |
| import React from 'react';  import ChartBar from './ChartBar';  import './Chart.css';  const Chart = props => {  return <div className="chart">  {props.dataPoints.map(**dataPoint => <ChartBar />**)}  </div>  };  export default Chart; |

And then of course we want to pass some data into the ChartBar to control how it will be rendered, so which value will be rendered in the ChartBar component. And for that, you of course want to extract some data from the incoming dataPoints.

|  |
| --- |
| src/components/Chart/Chart.js |
| import React from 'react';  import ChartBar from './ChartBar';  import './Chart.css';  const Chart = props => {  return <div className="chart">  {props.dataPoints.map(**dataPoint** => <ChartBar />)}  </div>  };  export default Chart; |

Now up to this point, we're never using the Chart component, so we as the creator of this component can define which kind of data we expect to extract there in the future. And for example, we could say that the ChartBar component should receive a value prop

|  |
| --- |
| src/components/Chart/Chart.js |
| import React from 'react';  import ChartBar from './ChartBar';  import './Chart.css';  const Chart = props => {  return <div className="chart">  {props.dataPoints.map(dataPoint => <ChartBar **value={}** />)}  </div>  };  export default Chart; |

and there we want to pass dataPoint.value.

|  |
| --- |
| src/components/Chart/Chart.js |
| import React from 'react';  import ChartBar from './ChartBar';  import './Chart.css';  const Chart = props => {  return <div className="chart">  {props.dataPoints.map(dataPoint => <ChartBar value={**dataPoint.value**} />)}  </div>  };  export default Chart; |

Of course, that will require that in the ChartBar we then read this prop, and that when we define the data points later, that every dataPoint has a value property. So, that dataPoint is an object which has a value property. That's what we're basically defining now since that is how we want to use that Chart component, and the ChartBar component in there.

|  |
| --- |
| src/components/Chart/Chart.js |
| import React from 'react';  import ChartBar from './ChartBar';  import './Chart.css';  const Chart = props => {  return <div className="chart">  {props.dataPoints.map(dataPoint => <ChartBar **value**={dataPoint.value} />)}  </div>  };  export default Chart; |

Now, I also want to make sure that every ChartBar plots the value in relation to the maximum value in the entire chart. Therefore, I also want to pass in a maxValue property, and that is

|  |
| --- |
| src/components/Chart/Chart.js |
| import React from 'react';  import ChartBar from './ChartBar';  import './Chart.css';  const Chart = props => {  return <div className="chart">  {props.dataPoints.map(dataPoint => <ChartBar value={dataPoint.value} **maxValue={}** />)}  </div>  };  export default Chart; |

currently null let's say. That's not data which we extract from the dataPoint because that is a unique value which is the same for all ChartBars in a given chart. So, we will need to derive this maxValue, which we will do later.

|  |
| --- |
| src/components/Chart/Chart.js |
| import React from "react";  import ChartBar from "./ChartBar";  import "./Chart.css";  const Chart = (props) => {  return (  <div className="chart">  {props.dataPoints.map((dataPoint) => (  <ChartBar value={dataPoint.value} maxValue={**null**} />  ))}  </div>  );  };  export default Chart; |

We also probably want to have a label, though. For example, in this case to have a label of January, February, March, April, and so on. And, therefore, I will add a label prop

|  |
| --- |
| src/components/Chart/Chart.js |
| import React from "react";  import ChartBar from "./ChartBar";  import "./Chart.css";  const Chart = (props) => {  return (  <div className="chart">  {props.dataPoints.map((dataPoint) => (  <ChartBar value={dataPoint.value} maxValue={null} **label={}** />  ))}  </div>  );  };  export default Chart; |

and pass in dataPoint.label.

|  |
| --- |
| src/components/Chart/Chart.js |
| import React from "react";  import ChartBar from "./ChartBar";  import "./Chart.css";  const Chart = (props) => {  return (  <div className="chart">  {props.dataPoints.map((dataPoint) => (  <ChartBar value={dataPoint.value} maxValue={null} label={**dataPoint.label**} />  ))}  </div>  );  };  export default Chart; |

And since we're mapping this, since we're outputting a list here, we should also add a key because you learned that this special key prop helps react render these list items efficiently.

|  |
| --- |
| src/components/Chart/Chart.js |
| import React from "react";  import ChartBar from "./ChartBar";  import "./Chart.css";  const Chart = (props) => {  return (  <div className="chart">  {props.dataPoints.map((dataPoint) => (  <ChartBar **key={}** value={dataPoint.value} maxValue={null} label={dataPoint.label} />  ))}  </div>  );  };  export default Chart; |

And, therefore, I do expect that every dataPoint also has a unique id

|  |
| --- |
| src/components/Chart/Chart.js |
| import React from "react";  import ChartBar from "./ChartBar";  import "./Chart.css";  const Chart = (props) => {  return (  <div className="chart">  {props.dataPoints.map((dataPoint) => (  <ChartBar key={**dataPoint.id**} value={dataPoint.value} maxValue={null} label={dataPoint.label} />  ))}  </div>  );  };  export default Chart; |

or maybe we actually use the label so that we say that the label should be unique. Every ChartBar has its own unique label, and therefore, we can also use the label as a unique identifier for this special key prop here.

|  |
| --- |
| src/components/Chart/Chart.js |
| import React from "react";  import ChartBar from "./ChartBar";  import "./Chart.css";  const Chart = (props) => {  return (  <div className="chart">  {props.dataPoints.map((dataPoint) => (  <ChartBar key={**dataPoint.label**} value={dataPoint.value} maxValue={null} label={dataPoint.label} />  ))}  </div>  );  };  export default Chart; |

### 70. Adding Dynamic Styles

Now, to build the ChartBar component, we will start with our basic component setup. We will need props because we are already passing props to ChartBar in "Chart.js".

|  |
| --- |
| src/components/Chart/ChartBar.js |
| import React from "react";  import './ChartBar.css';  const ChartBar = (props) => {};  export default ChartBar; |

We add return because we will return our JSX code here:

|  |
| --- |
| src/components/Chart/ChartBar.js |
| import React from "react";  import './ChartBar.css';  const ChartBar = (props) => {  **return**  };  export default ChartBar; |

Now, for this JSX code in the ChartBar which we return, I want to have a div here with a className of "chart-bar" since that's one of the CSS styles defined in "ChartBar.css".

|  |
| --- |
| src/components/Chart/ChartBar.js |
| import React from "react";  import "./ChartBar.css";  const ChartBar = (props) => {  return **<div className="chart-bar"></div>**  };  export default ChartBar; |

Within the div with the className of "chart-bar", we will have a nested div with a clasName of "chart-bar\_\_inner"

|  |
| --- |
| src/components/Chart/ChartBar.js |
| import React from "react";  import "./ChartBar.css";  const ChartBar = (props) => {  return (  <div className="chart-bar">  **<div className="chart-bar\_\_inner"></div>**  </div>  );  };  export default ChartBar; |

These divs and CSS classes are simply required to make the provided styles work. Within the div with the className of "chart-bar\_\_inner", we will have a nested div with a clasName of "chart-bar\_\_fill".

|  |
| --- |
| src/components/Chart/ChartBar.js |
| import React from "react";  import "./ChartBar.css";  const ChartBar = (props) => {  return (  <div className="chart-bar">  <div className="chart-bar\_\_inner">  **<div className="chart-bar\_\_fill"></div>**  </div>  </div>  );  };  export default ChartBar; |

Next to the div with the className of "chart-bar\_\_inner", I'll add another div with the className of "chart-bar\_\_label".

|  |
| --- |
| src/components/Chart/ChartBar.js |
| import React from "react";  import "./ChartBar.css";  const ChartBar = (props) => {  return (  <div className="chart-bar">  <div className="chart-bar\_\_inner">  <div className="chart-bar\_\_fill"></div>  </div>  **<div className="chart-bar\_\_label"></div>**  </div>  );  };  export default ChartBar; |

The structure in the ChartBar component is a bunch of divs that will then be responsible for rendering this chart correctly.

With just this, we wouldn't see much.

|  |
| --- |
| src/components/Chart/ChartBar.js |
| import React from "react";  import "./ChartBar.css";  const ChartBar = (props) => {  return (  <div className="chart-bar">  <div className="chart-bar\_\_inner">  <div className="chart-bar\_\_fill"></div>  </div>  <div className="chart-bar\_\_label"></div>  </div>  );  };  export default ChartBar; |

We of course want to output the label, which we receive on the label prop,

|  |
| --- |
| src/components/Chart/Chart.js |
| import React from "react";  import ChartBar from "./ChartBar";  import "./Chart.css";  const Chart = (props) => {  return (  <div className="chart">  {props.dataPoints.map((dataPoint) => (  <ChartBar  key={dataPoint.label}  value={dataPoint.value}  maxValue={null}  **label**={dataPoint.label}  />  ))}  </div>  );  };  export default Chart; |

Which is why we add this div here:

|  |
| --- |
| src/components/Chart/ChartBar.js |
| import React from "react";  import "./ChartBar.css";  const ChartBar = (props) => {  return (  <div className="chart-bar">  <div className="chart-bar\_\_inner">  <div className="chart-bar\_\_fill"></div>  </div>  **<div className="chart-bar\_\_label"></div>**  </div>  );  };  export default ChartBar; |

Here, we can dynamically output props.label, so that the label is visible.

|  |
| --- |
| src/components/Chart/ChartBar.js |
| import React from "react";  import "./ChartBar.css";  const ChartBar = (props) => {  return (  <div className="chart-bar">  <div className="chart-bar\_\_inner">  <div className="chart-bar\_\_fill"></div>  </div>  <div className="chart-bar\_\_label">**{props.label}**</div>  </div>  );  };  export default ChartBar; |

But we also will need to change this div with the className "chart-bar\_\_fill". This div is required to basically remember by how much this ChartBar will be filled. And this "chart-bar\_\_fill" CSS class will for example, define the background-color for that. But one important piece of information is missing in this CSS class, and that will be the height of this filled ChartBar.

|  |
| --- |
| src/components/Chart/ChartBar.js |
| import React from "react";  import "./ChartBar.css";  const ChartBar = (props) => {  return (  <div className="chart-bar">  <div className="chart-bar\_\_inner">  **<div className="chart-bar\_\_fill"></div>**  </div>  <div className="chart-bar\_\_label">{props.label}</div>  </div>  );  };  export default ChartBar; |

The overall ChartBar has a predefined height, which is specified in "ChartBar.css" in ".chart-bar\_\_inner" with a height of 100% of the parent container, which is ".chart" specified in "Chart.css" with a height of 10 rem. But, of course, how much we fill that bar depends on the data that we're receiving. So, it depends on the value (bolded in the below code snippet) because we basically want to fill our ChartBar by putting the value in relation with the maxValue. So that if the maxValue for the entire chart is 100, and the value for a given ChartBar is 50, we fill that ChartBar by 50%.

|  |
| --- |
| src/components/Chart/Chart.js |
| import React from "react";  import ChartBar from "./ChartBar";  import "./Chart.css";  const Chart = (props) => {  return (  <div className="chart">  {props.dataPoints.map((dataPoint) => (  <ChartBar  key={dataPoint.label}  **value={dataPoint.value}**  maxValue={null}  label={dataPoint.label}  />  ))}  </div>  );  };  export default Chart; |

Therefore, in the ChartBar component we need to calculate by how much this specific ChartBar instance should be filled. For this, we can create a variable called barFillHeight in "ChartBar.js", and initially I'll set this to 0% as a String because this will be assigned to a CSS style.

|  |
| --- |
| src/components/Chart/ChartBar.js |
| import React from "react";  import "./ChartBar.css";  const ChartBar = (props) => {  **let barFillHeight = "0%";**  return (  <div className="chart-bar">  <div className="chart-bar\_\_inner">  <div className="chart-bar\_\_fill"></div>  </div>  <div className="chart-bar\_\_label">{props.label}</div>  </div>  );  };  export default ChartBar; |

We then want to check if we get a maxValue greater than 0. We could have 0 later if we filter for a month that has no expenses. So we want to check if we have a maxValue greater than 0.

|  |
| --- |
| src/components/Chart/ChartBar.js |
| import React from "react";  import "./ChartBar.css";  const ChartBar = (props) => {  let barFillHeight = "0%";  **if (props.maxValue > 0) {**  **}**  return (  <div className="chart-bar">  <div className="chart-bar\_\_inner">  <div className="chart-bar\_\_fill"></div>  </div>  <div className="chart-bar\_\_label">{props.label}</div>  </div>  );  };  export default ChartBar; |

If we do have props.maxValue greater than 0, I want to set barFillHeight equal to Math.round((props.value / props.maxValue) \* 100), which will give us the percentage between 0 and 100 by which this bar should be filled. We will also add + '%' at the end because we want to convert to a string.

|  |
| --- |
| src/components/Chart/ChartBar.js |
| import React from "react";  import "./ChartBar.css";  const ChartBar = (props) => {  let barFillHeight = "0%";  if (props.maxValue > 0) {  **barFillHeight = Math.round((props.value / props.maxValue) \* 100) + '%';**  }  return (  <div className="chart-bar">  <div className="chart-bar\_\_inner">  <div className="chart-bar\_\_fill"></div>  </div>  <div className="chart-bar\_\_label">{props.label}</div>  </div>  );  };  export default ChartBar; |

And now we want to set barFillHeight as the CSS style height for the div with the className 'chart-bar\_\_fill'.

|  |
| --- |
| src/components/Chart/ChartBar.js |
| import React from "react";  import "./ChartBar.css";  const ChartBar = (props) => {  let barFillHeight = "0%";  if (props.maxValue > 0) {  barFillHeight = Math.round((props.value / props.maxValue) \* 100) + '%';  }  return (  <div className="chart-bar">  <div className="chart-bar\_\_inner">  **<div className="chart-bar\_\_fill"></div>**  </div>  <div className="chart-bar\_\_label">{props.label}</div>  </div>  );  };  export default ChartBar; |

We are going to do something that we haven't done thus far in this course. We're going to set the style of an element dynamically. That can be done by adding the style prop

|  |
| --- |
| src/components/Chart/ChartBar.js |
| import React from "react";  import "./ChartBar.css";  const ChartBar = (props) => {  let barFillHeight = "0%";  if (props.maxValue > 0) {  barFillHeight = Math.round((props.value / props.maxValue) \* 100) + '%';  }  return (  <div className="chart-bar">  <div className="chart-bar\_\_inner">  <div className="chart-bar\_\_fill" **style**></div>  </div>  <div className="chart-bar\_\_label">{props.label}</div>  </div>  );  };  export default ChartBar; |

which is of course a default HTML attribute as well but actually that style prop, that style attribute, works a little differently when building a React application. The value we set for style should be dynamic because it should be our derived barFillHeight.

|  |
| --- |
| src/components/Chart/ChartBar.js |
| import React from "react";  import "./ChartBar.css";  const ChartBar = (props) => {  let barFillHeight = "0%";  if (props.maxValue > 0) {  barFillHeight = Math.round((props.value / props.maxValue) \* 100) + '%';  }  return (  <div className="chart-bar">  <div className="chart-bar\_\_inner">  <div className="chart-bar\_\_fill" style=**{}**></div>  </div>  <div className="chart-bar\_\_label">{props.label}</div>  </div>  );  };  export default ChartBar; |

You then don't generate a dynamic string where you set something like height 10%, but instead style wants an object. So, here we don't have a special double curly brace syntax but instead we have

|  |
| --- |
| src/components/Chart/ChartBar.js |
| import React from "react";  import "./ChartBar.css";  const ChartBar = (props) => {  let barFillHeight = "0%";  if (props.maxValue > 0) {  barFillHeight = Math.round((props.value / props.maxValue) \* 100) + '%';  }  return (  <div className="chart-bar">  <div className="chart-bar\_\_inner">  <div className="chart-bar\_\_fill" style={**{}**}></div>  </div>  <div className="chart-bar\_\_label">{props.label}</div>  </div>  );  };  export default ChartBar; |

a single curly brace syntax for outputting something dynamically,

|  |
| --- |
| src/components/Chart/ChartBar.js |
| import React from "react";  import "./ChartBar.css";  const ChartBar = (props) => {  let barFillHeight = "0%";  if (props.maxValue > 0) {  barFillHeight = Math.round((props.value / props.maxValue) \* 100) + '%';  }  return (  <div className="chart-bar">  <div className="chart-bar\_\_inner">  <div className="chart-bar\_\_fill" style=**{**{}**}**></div>  </div>  <div className="chart-bar\_\_label">{props.label}</div>  </div>  );  };  export default ChartBar; |

but then the dynamic value is a JavaScript object, which is also created with curly braces.

|  |
| --- |
| src/components/Chart/ChartBar.js |
| import React from "react";  import "./ChartBar.css";  const ChartBar = (props) => {  let barFillHeight = "0%";  if (props.maxValue > 0) {  barFillHeight = Math.round((props.value / props.maxValue) \* 100) + '%';  }  return (  <div className="chart-bar">  <div className="chart-bar\_\_inner">  <div className="chart-bar\_\_fill" style={**{}**}></div>  </div>  <div className="chart-bar\_\_label">{props.label}</div>  </div>  );  };  export default ChartBar; |

That's why overall we have the double curly braces, but it's no special style syntax. The only special thing here is that style wants a JavaScript object as a value.

|  |
| --- |
| src/components/Chart/ChartBar.js |
| import React from "react";  import "./ChartBar.css";  const ChartBar = (props) => {  let barFillHeight = "0%";  if (props.maxValue > 0) {  barFillHeight = Math.round((props.value / props.maxValue) \* 100) + '%';  }  return (  <div className="chart-bar">  <div className="chart-bar\_\_inner">  <div className="chart-bar\_\_fill" style=**{{}}**></div>  </div>  <div className="chart-bar\_\_label">{props.label}</div>  </div>  );  };  export default ChartBar; |

and then in this JS object you should then use the CSS property names as key names as properties and the values as values.

|  |
| --- |
| src/components/Chart/ChartBar.js |
| import React from "react";  import "./ChartBar.css";  const ChartBar = (props) => {  let barFillHeight = "0%";  if (props.maxValue > 0) {  barFillHeight = Math.round((props.value / props.maxValue) \* 100) + '%';  }  return (  <div className="chart-bar">  <div className="chart-bar\_\_inner">  <div className="chart-bar\_\_fill" style={**{}**}></div>  </div>  <div className="chart-bar\_\_label">{props.label}</div>  </div>  );  };  export default ChartBar; |

For example, here we can then set style={{'background-color': 'red'}}

|  |
| --- |
| src/components/Chart/ChartBar.js |
| import React from "react";  import "./ChartBar.css";  const ChartBar = (props) => {  let barFillHeight = "0%";  if (props.maxValue > 0) {  barFillHeight = Math.round((props.value / props.maxValue) \* 100) + "%";  }  return (  <div className="chart-bar">  <div className="chart-bar\_\_inner">  <div  className="chart-bar\_\_fill"  style={{ **"background-color": "red"** }}  ></div>  </div>  <div className="chart-bar\_\_label">{props.label}</div>  </div>  );  };  export default ChartBar; |

or in our case we can set height to barFillHeight

|  |
| --- |
| src/components/Chart/ChartBar.js |
| import React from "react";  import "./ChartBar.css";  const ChartBar = (props) => {  let barFillHeight = "0%";  if (props.maxValue > 0) {  barFillHeight = Math.round((props.value / props.maxValue) \* 100) + "%";  }  return (  <div className="chart-bar">  <div className="chart-bar\_\_inner">  <div  className="chart-bar\_\_fill"  style={{ **height: barFillHeight** }}  ></div>  </div>  <div className="chart-bar\_\_label">{props.label}</div>  </div>  );  };  export default ChartBar; |

As a side note, if you have a style key like background-color with a dash, you need the quotes around the property name / key because otherwise it would be an invalid property name.

|  |
| --- |
| src/components/Chart/ChartBar.js |
| import React from "react";  import "./ChartBar.css";  const ChartBar = (props) => {  let barFillHeight = "0%";  if (props.maxValue > 0) {  barFillHeight = Math.round((props.value / props.maxValue) \* 100) + "%";  }  return (  <div className="chart-bar">  <div className="chart-bar\_\_inner">  <div  className="chart-bar\_\_fill"  style={{ height: barFillHeight, **"background-color"**: "red" }}  ></div>  </div>  <div className="chart-bar\_\_label">{props.label}</div>  </div>  );  };  export default ChartBar; |

Or, better would be to use the camel case version of that property name i.e. backgroundColor. We don't need the backgroundColor in our app, but that is how you would target such style property names with dashes inside of them if you are setting the style prop on an element.

|  |
| --- |
| src/components/Chart/ChartBar.js |
| import React from "react";  import "./ChartBar.css";  const ChartBar = (props) => {  let barFillHeight = "0%";  if (props.maxValue > 0) {  barFillHeight = Math.round((props.value / props.maxValue) \* 100) + "%";  }  return (  <div className="chart-bar">  <div className="chart-bar\_\_inner">  <div  className="chart-bar\_\_fill"  style={{ height: barFillHeight, **backgroundColor**: "red" }}  ></div>  </div>  <div className="chart-bar\_\_label">{props.label}</div>  </div>  );  };  export default ChartBar; |

Because we don't need the backgroundColor property in our app, we will get rid of it.

|  |
| --- |
| src/components/Chart/ChartBar.js |
| import React from "react";  import "./ChartBar.css";  const ChartBar = (props) => {  let barFillHeight = "0%";  if (props.maxValue > 0) {  barFillHeight = Math.round((props.value / props.maxValue) \* 100) + "%";  }  return (  <div className="chart-bar">  <div className="chart-bar\_\_inner">  <div  className="chart-bar\_\_fill"  style={{ height: barFillHeight }}  ></div>  </div>  <div className="chart-bar\_\_label">{props.label}</div>  </div>  );  };  export default ChartBar; |

With that, we have now finished the ChartBar component. Now, we just need to use the Chart and pass in the dataPoints.

### 71. Wrap Up & Next Steps

So now to use the Chart and pass in the dataPoints, I will add a new component file in the "Expenses" component folder and will name it "ExpensesChart.js". In the "ExpenseChart.js" file we don't even need a CSS file, but we will do our basic functional component setup.

|  |
| --- |
| src/components/Expenses/ExpensesChart.js |
| import React from "react";  const ExpensesChart = (props) => {};  export default ExpensesChart; |

Inside of the ExpenseChart component, the overall goal here is to return the Chart.

|  |
| --- |
| src/components/Expenses/ExpensesChart.js |
| import React from "react";  import Chart from "../Chart/Chart";  const ExpensesChart = (props) => {  **return <Chart />;**  };  export default ExpensesChart; |

We now need to define the dataPoints that are passed into the Chart because there we are referring to this dataPoints prop.

|  |
| --- |
| src/components/Chart/Chart.js |
| import React from "react";  import ChartBar from "./ChartBar";  import "./Chart.css";  const Chart = (props) => {  return (  <div className="chart">  {props.**dataPoints**.map((dataPoint) => (  <ChartBar  key={dataPoint.label}  value={dataPoint.value}  maxValue={null}  label={dataPoint.label}  />  ))}  </div>  );  };  export default Chart; |

And to set up the dataPoints in ExpenseChart, we will create a new constant, and name it chartDataPoints, which will be an array, and in this array I want to have a bunch of objects because we expect every dataPoint to be an object, as shown in the "Chart.js" file when we map through dataPoints, we then access values like label and value on the given datapoint. Therefore, we want to have objects, and these objects should have a label and a value key. The value is simply 0 initially for every datapoint, and the label is an abbreviated month name such as 'Jan' for January.

|  |
| --- |
| src/components/Expenses/ExpensesChart.js |
| import React from "react";  import Chart from "../Chart/Chart";  const ExpensesChart = (props) => {  **const chartDataPoints = [**  **{ label: "Jan", value: 0 },**  **{ label: "Feb", value: 0 },**  **{ label: "Mar", value: 0 },**  **{ label: "Apr", value: 0 },**  **{ label: "May", value: 0 },**  **{ label: "Jun", value: 0 },**  **{ label: "Jul", value: 0 },**  **{ label: "Aug", value: 0 },**  **{ label: "Sep", value: 0 },**  **{ label: "Oct", value: 0 },**  **{ label: "Nov", value: 0 },**  **{ label: "Dec", value: 0 },**  **];**  return <Chart />;  };  export default ExpensesChart; |

We don't want to have 0 for every data point. Instead, now we want to have a look at our filteredExpenses and make sure that we basically go through all the expenses for a selected year and that we then sum up the expenses for all the different months, and we assign them in the chartDataPoints object to our data points. So therefore, I expect to get filteredExpenses as a prop on my ExpenseChart component because we will use ExpenseChart in the "Expenses.js" file, and in that file we have the filteredExpenses constant.

|  |
| --- |
| src/components/Expenses/Expenses.js |
| import React, { useState } from 'react';  import Card from '../UI/Card';  import ExpensesFilter from './ExpensesFilter';  import ExpensesList from './ExpensesList';  import './Expenses.css';  const Expenses = (props) => {  const [filteredYear, setFilteredYear] = useState('2020');  const filterChangeHandler = (selectedYear) => {  setFilteredYear(selectedYear);  };  **const filteredExpenses = props.items.filter((expense) => {**  **return expense.date.getFullYear().toString() === filteredYear;**  **});**  return (  <div>  <Card className='expenses'>  <ExpensesFilter  selected={filteredYear}  onChangeFilter={filterChangeHandler}  />  <ExpensesList items={filteredExpenses} />  </Card>  </div>  );  };  export default Expenses; |

So here in ExpensesChart, we just expect to get that list of expenses. In ExpensesChart we can add a for loop where we loop through all our expenses, which we get via props.

|  |
| --- |
| src/components/Expenses/ExpensesChart.js |
| import React from "react";  import Chart from "../Chart/Chart";  const ExpensesChart = (props) => {  const chartDataPoints = [  { label: "Jan", value: 0 },  { label: "Feb", value: 0 },  { label: "Mar", value: 0 },  { label: "Apr", value: 0 },  { label: "May", value: 0 },  { label: "Jun", value: 0 },  { label: "Jul", value: 0 },  { label: "Aug", value: 0 },  { label: "Sep", value: 0 },  { label: "Oct", value: 0 },  { label: "Nov", value: 0 },  { label: "Dec", value: 0 },  ];  **for (const expense of props.expenses) {**    **}**  return <Chart />;  };  export default ExpensesChart; |

And then we want to have a look at every expense, get the month of that expense, and update the value of the appropriate data point of chartDataPoints by the expense amount. So inside of the loop we define a constant called expenseMonth and set it to expense.date.getMonth(); Remember that date is a date object and there we have the built in getMonth() method, which returns us that month starting at 0, though, so January is 0. We can then use this month later to pick the correct data point from chartDataPoints since January has the index 0 in the chartDataPoints array as well.

|  |
| --- |
| src/components/Expenses/ExpensesChart.js |
| import React from "react";  import Chart from "../Chart/Chart";  const ExpensesChart = (props) => {  const chartDataPoints = [  { label: "Jan", value: 0 },  { label: "Feb", value: 0 },  { label: "Mar", value: 0 },  { label: "Apr", value: 0 },  { label: "May", value: 0 },  { label: "Jun", value: 0 },  { label: "Jul", value: 0 },  { label: "Aug", value: 0 },  { label: "Sep", value: 0 },  { label: "Oct", value: 0 },  { label: "Nov", value: 0 },  { label: "Dec", value: 0 },  ];  for (const expense of props.expenses) {  **const expenseMonth = expense.date.getMonth();**  }  return <Chart />;  };  export default ExpensesChart; |

So, we can basically use the expenseMonth as an index in the chartDataPoints array. I will reach out to my chartDataPoints array for the expenseMonth. Again, the expenseMonth starts at 0 and ends at 11, which is exactly the case for the index of the data points in the chartDataPoints array. We want to update the value of the selected data point by adding something to it with the plus equals shortcut operator. I want to add expense.amount to it, so that we increase the value of a given month by that expense.amount. And we go through all of the expenses to sum up all the expenses for the different months and assign values to the appropriate months, to the appropriate data points in the chartDataPoints array. So that after the for loop, we still have the chartDataPoints array, but now the values will not be 0 anymore but instead they will have the summed-up values for the given expenseMonth.

|  |
| --- |
| src/components/Expenses/ExpensesChart.js |
| import React from "react";  import Chart from "../Chart/Chart";  const ExpensesChart = (props) => {  const chartDataPoints = [  { label: "Jan", value: 0 },  { label: "Feb", value: 0 },  { label: "Mar", value: 0 },  { label: "Apr", value: 0 },  { label: "May", value: 0 },  { label: "Jun", value: 0 },  { label: "Jul", value: 0 },  { label: "Aug", value: 0 },  { label: "Sep", value: 0 },  { label: "Oct", value: 0 },  { label: "Nov", value: 0 },  { label: "Dec", value: 0 },  ];  for (const expense of props.expenses) {  const expenseMonth = expense.date.getMonth(); // Starting at 0 => January => 0  **chartDataPoints[expenseMonth].value += expense.amount;**  }  return <Chart />;  };  export default ExpensesChart; |

And now we can pass these dataPoints to the Chart.

|  |
| --- |
| src/components/Expenses/ExpensesChart.js |
| import React from "react";  import Chart from "../Chart/Chart";  const ExpensesChart = (props) => {  const chartDataPoints = [  { label: "Jan", value: 0 },  { label: "Feb", value: 0 },  { label: "Mar", value: 0 },  { label: "Apr", value: 0 },  { label: "May", value: 0 },  { label: "Jun", value: 0 },  { label: "Jul", value: 0 },  { label: "Aug", value: 0 },  { label: "Sep", value: 0 },  { label: "Oct", value: 0 },  { label: "Nov", value: 0 },  { label: "Dec", value: 0 },  ];  for (const expense of props.expenses) {  const expenseMonth = expense.date.getMonth(); // Starting at 0 => January => 0  chartDataPoints[expenseMonth].value += expense.amount;  }  return <Chart **dataPoints** />;  };  export default ExpensesChart; |

There we set the dataPoints prop since that's the prop we're expecting there.

|  |
| --- |
| src/components/Chart/Chart.js |
| import React from "react";  import ChartBar from "./ChartBar";  import "./Chart.css";  const Chart = (props) => {  return (  <div className="chart">  {props.**dataPoints**.map((dataPoint) => (  <ChartBar  key={dataPoint.label}  value={dataPoint.value}  maxValue={null}  label={dataPoint.label}  />  ))}  </div>  );  };  export default Chart; |

And inside of ExpenseChart within the call to Chart, we set the dataPoints prop equal to our updated chartDataPoints.

|  |
| --- |
| src/components/Expenses/ExpensesChart.js |
| import React from "react";  import Chart from "../Chart/Chart";  const ExpensesChart = (props) => {  const chartDataPoints = [  { label: "Jan", value: 0 },  { label: "Feb", value: 0 },  { label: "Mar", value: 0 },  { label: "Apr", value: 0 },  { label: "May", value: 0 },  { label: "Jun", value: 0 },  { label: "Jul", value: 0 },  { label: "Aug", value: 0 },  { label: "Sep", value: 0 },  { label: "Oct", value: 0 },  { label: "Nov", value: 0 },  { label: "Dec", value: 0 },  ];  for (const expense of props.expenses) {  const expenseMonth = expense.date.getMonth(); // Starting at 0 => January => 0  chartDataPoints[expenseMonth].value += expense.amount;  }  return <Chart dataPoints=**{chartDataPoints}** />;  };  export default ExpensesChart; |

We still want to go back to our Chart component, and there we need to calculate the total maxValue. So, we want to have a look at all of the months and find the biggest value across all months because that's the maximum value that should be represented in the chart. To do this, we can add a new constant called totalMaximum, and here we can use Math.max() to find the maximum value, but actually max wants a number of arguments like this, for which it then returns the biggest number.

|  |
| --- |
| src/components/Chart/Chart.js |
| import React from "react";  import ChartBar from "./ChartBar";  import "./Chart.css";  const Chart = (props) => {  **const totalMaximum = Math.max(1, 2, 3);**  return (  <div className="chart">  {props.dataPoints.map((dataPoint) => (  <ChartBar  key={dataPoint.label}  value={dataPoint.value}  maxValue={null}  label={dataPoint.label}  />  ))}  </div>  );  };  export default Chart; |

Here we have an array, though, and not an array of numbers but an array of objects, and we're just interested in a specific property of that object.

|  |
| --- |
| src/components/Chart/Chart.js |
| import React from "react";  import ChartBar from "./ChartBar";  import "./Chart.css";  const Chart = (props) => {  **const totalMaximum = Math.max();**  return (  <div className="chart">  {props.dataPoints.map((dataPoint) => (  <ChartBar  key={dataPoint.label}  value={dataPoint.value}  maxValue={null}  label={dataPoint.label}  />  ))}  </div>  );  };  export default Chart; |

Nonetheless, we can work around this by creating a constant variable called dataPointValues and setting it to props.dataPoints.map() to call the map() method, but now we won't map the data points into JSX elements, but instead simply transform them from objects to numbers, so that for every dataPoint, we in the end just return dataPoint.value, so that means that we transform a datapoint object to just the number, the number stored in dataPoint.value. And therefore, map on dataPoints will return a brand-new array which is just an array of numbers for all of the dataPoints that we are getting. So in our case for the 12 months, we will have an array of 12 values.

|  |
| --- |
| src/components/Chart/Chart.js |
| import React from "react";  import ChartBar from "./ChartBar";  import "./Chart.css";  const Chart = (props) => {  **const dataPointValues = props.dataPoints.map(dataPoint => dataPoint.value);**  const totalMaximum = Math.max();  return (  <div className="chart">  {props.dataPoints.map((dataPoint) => (  <ChartBar  key={dataPoint.label}  value={dataPoint.value}  maxValue={null}  label={dataPoint.label}  />  ))}  </div>  );  };  export default Chart; |

And now it's the dataPointValues where we want to find the maximum. But since max wants a list of arguments instead of an array and since dataPointValues is still an array, we can use the spread operator to pull out all the array elements and add them as standalone arguments to this max method. So now the max method will receive 12 arguments, which are these 12 values from our array pulled out by this spread operator.

|  |
| --- |
| src/components/Chart/Chart.js |
| import React from "react";  import ChartBar from "./ChartBar";  import "./Chart.css";  const Chart = (props) => {  const dataPointValues = props.dataPoints.map((dataPoint) => dataPoint.value);  const totalMaximum = Math.max(**...dataPointValues**);  return (  <div className="chart">  {props.dataPoints.map((dataPoint) => (  <ChartBar  key={dataPoint.label}  value={dataPoint.value}  maxValue={null}  label={dataPoint.label}  />  ))}  </div>  );  };  export default Chart; |

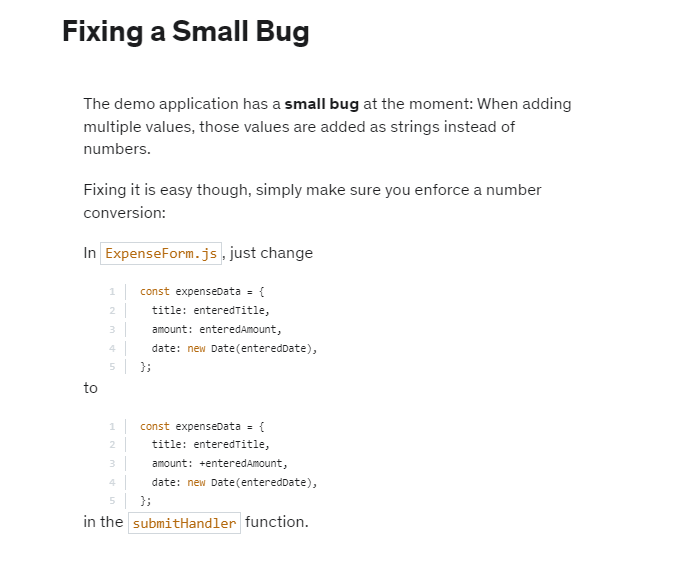
Now, we take totalMaximum and pass it in as the value of the maxValue prop.

|  |
| --- |
| src/components/Chart/Chart.js |
| import React from "react";  import ChartBar from "./ChartBar";  import "./Chart.css";  const Chart = (props) => {  const dataPointValues = props.dataPoints.map((dataPoint) => dataPoint.value);  const totalMaximum = Math.max(...dataPointValues);  return (  <div className="chart">  {props.dataPoints.map((dataPoint) => (  <ChartBar  key={dataPoint.label}  value={dataPoint.value}  maxValue={**totalMaximum**}  label={dataPoint.label}  />  ))}  </div>  );  };  export default Chart; |

We now just want to make sure that we use our ExpensesChart component, and we want to do that in our "Expenses.js" file. We render our ExpensesChart between the ExpensesFilter and the ExpensesList components. We also need to pass in the expenses prop because that is expected in the ExpensesChart component. We set expenses={filteredExpenses} because of course I only want to chart the currently filteredExpenses.

|  |
| --- |
| src/components/Expenses/Expenses.js |
| import React, { useState } from "react";  import Card from "../UI/Card";  import ExpensesFilter from "./ExpensesFilter";  import ExpensesList from "./ExpensesList";  import ExpensesChart from "./ExpensesChart";  import "./Expenses.css";  const Expenses = (props) => {  const [filteredYear, setFilteredYear] = useState("2020");  const filterChangeHandler = (selectedYear) => {  setFilteredYear(selectedYear);  };  const filteredExpenses = props.items.filter((expense) => {  return expense.date.getFullYear().toString() === filteredYear;  });  return (  <div>  <Card className="expenses">  <ExpensesFilter  selected={filteredYear}  onChangeFilter={filterChangeHandler}  />  **<ExpensesChart expenses={filteredExpenses} />**  <ExpensesList items={filteredExpenses} />  </Card>  </div>  );  };  export default Expenses; |

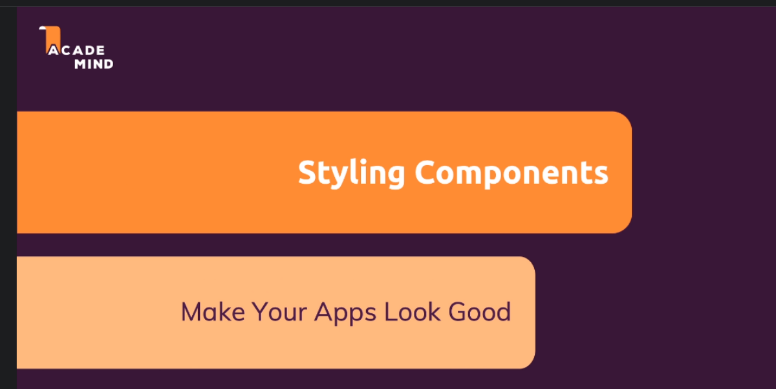
### 72. Fixing a Small Bug



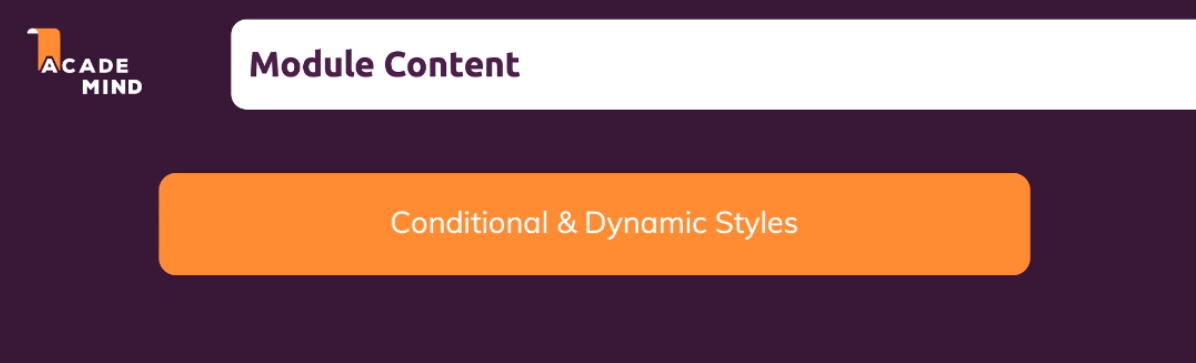
## Section 6: Styling React Components

### 74. Module Introduction

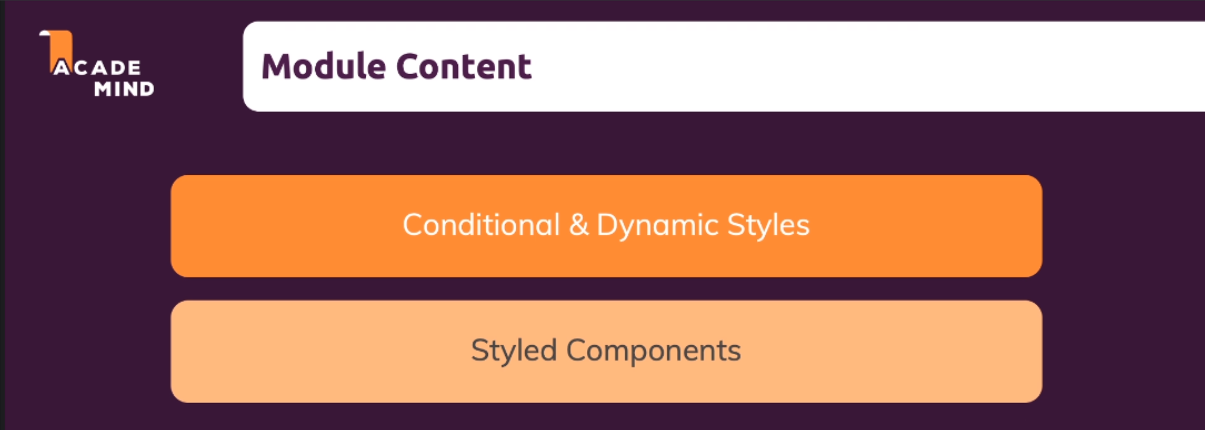
Building web apps with React is not just about composing components together and making sure that the logic works correctly. That's arguably the most important part, or at least it's the biggest reason for using React. But building React apps is also about styling those apps. You want to make sure that your applications look good, and styling is also an important part of building components. There are different techniques for setting styles dynamically and for styling components such that other components are not affected by as specific component's styles.



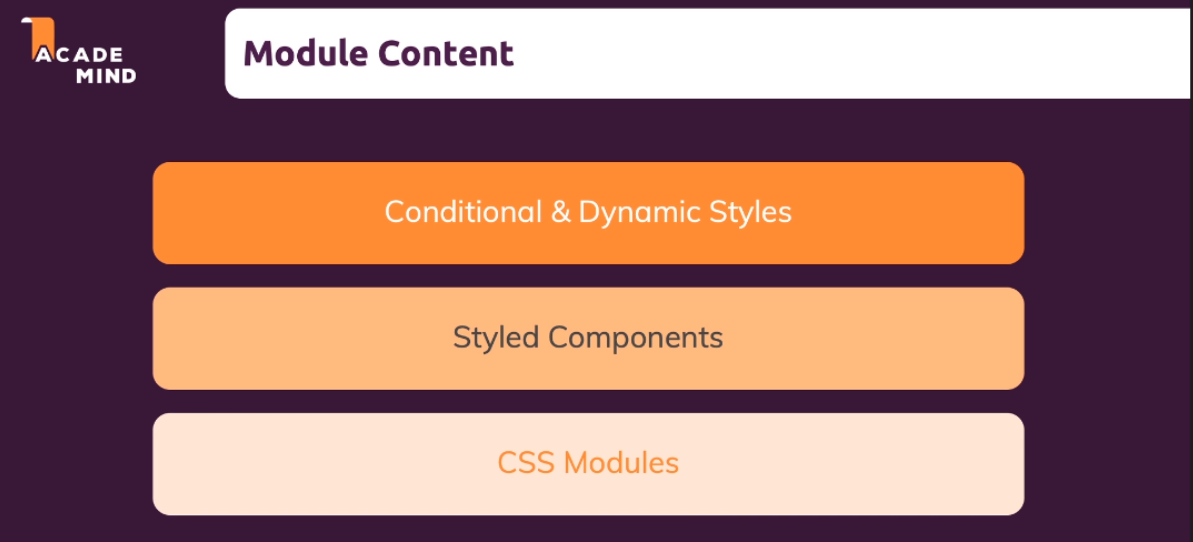
Therefore, in this module, we will dive into how you can setup styles in a conditional or dynamic way. So how you, for example, can switch certain styles based on some conditions. For example, if a user enters an invalid input and you want to color the input red. And then we'll dive into two popular approaches for scoping styles to components, so for setting up styles such that they only affect the component and are not added as global styles.



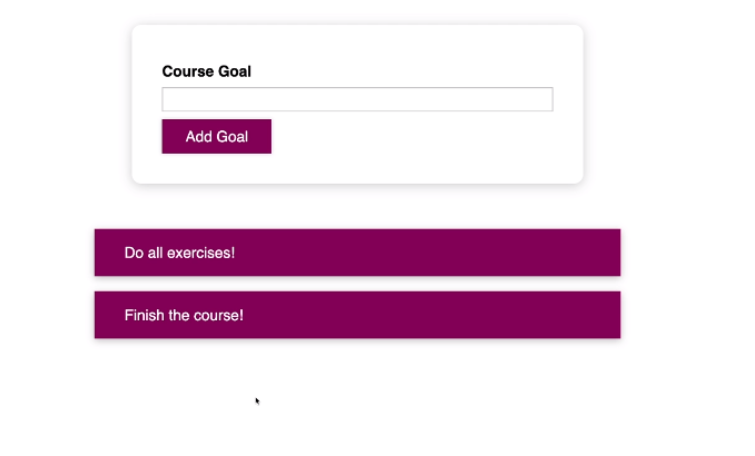
We'll have a look at Styled Components, which is a third-party library, a very popular one, which allows us to set up pre-styled components with their own scoped styles.



And we'll have a look at a concept called CSS modules. And you will learn what these different approaches are all about, how you do work with them, and how you could add them to your next project.



To dive into styling, we need a demo project, and to mix things up, we have a brand new project. You can basically add course goals to a list. You can delete items from the list by clicking on them.



This project includes only things that were covered over the last course sections like useState and functions which are bound to events. We’re also outputting lists of data dynamically by calling map.

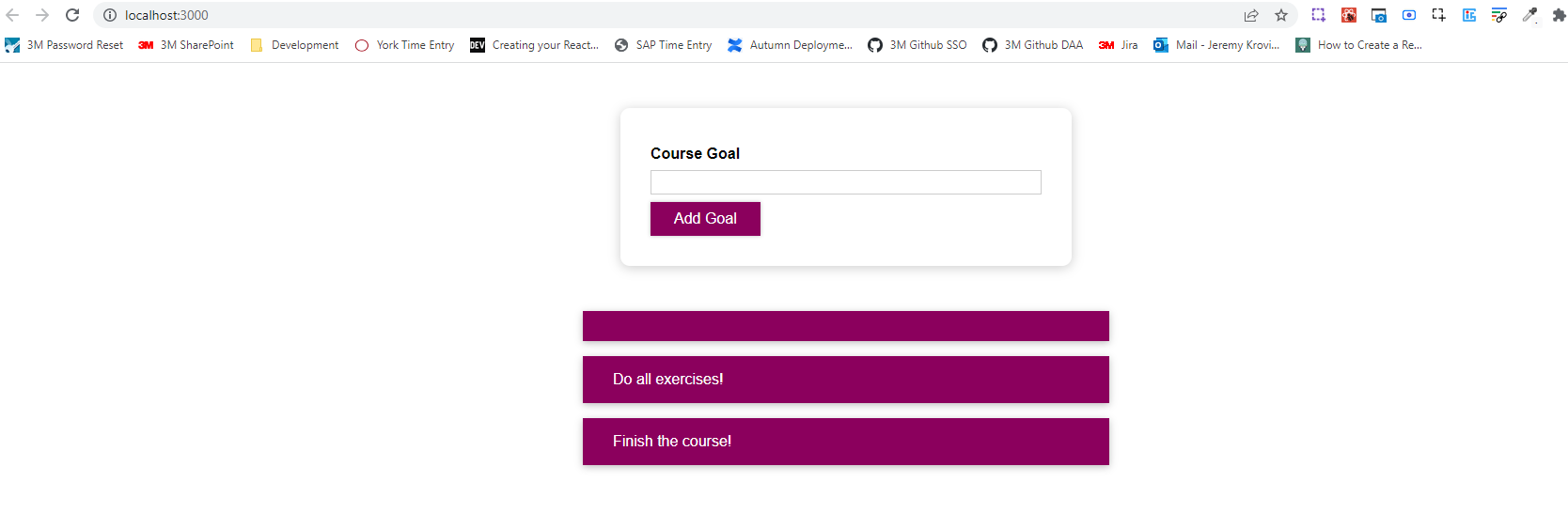
### 75. Setting Dynamic Inline Styles

We may want to change the way styles are applied because at the moment we are using regular CSS, with regular CSS selectors. Even though, we import './CourseGoalList.css'; into the CourseGoalList component, it is not scoped to that component. It would affect any element on the entire page which has a goal list, CSS class. So that's one thing we'll tackle, but we'll start with something different, with setting styles dynamically.

A good example can be found in the CourseInput component. There we are collecting some input by the user and then we have the "Add Goal" Button to add a goal.

|  |
| --- |
| src/components/CourseGoals/CourseInput/CourseInput.js |
| import React, { useState } from 'react';  import Button from '../../UI/Button/Button';  import './CourseInput.css';  const CourseInput = props => {  const [enteredValue, setEnteredValue] = useState('');  const goalInputChangeHandler = event => {  setEnteredValue(event.target.value);  };  const formSubmitHandler = event => {  event.preventDefault();  props.onAddGoal(enteredValue);  };  return (  <form onSubmit={formSubmitHandler}>  <div className="form-control">  <label>Course Goal</label>  **<input type="text" onChange={goalInputChangeHandler} />**  </div>  **<Button type="submit">Add Goal</Button>**  </form>  );  };  export default CourseInput; |

In the moment, we can press the "Add Goal" button even if we didn't enter anything, and the result is that we add an empty element. Now, in most applications we want to avoid this, and we don't just want to avoid this, we also want to give the user some feedback about the incorrect input. That's something we'll build now, and that's something we'll need to set styles dynamically.



So, in CourseInput component, it’s the formSubmitHandler function where I in the end want to check whether something valid was entered before we trigger that onAddGoal function. So that function we get on this onAddGoal prop. For this we can add a simple if statement here in the formSubmitHandler and check if the enteredValue, if we trim it, trim is a built in method that removes excess white space at the beginning or the end, and I use it here to rule out that the user just entered a bunch of blanks, and we check the length if the length is now equal to zero, we know that the input is essentially empty.

|  |
| --- |
| src/components/CourseGoals/CourseInput/CourseInput.js |
| import React, { useState } from "react";  import Button from "../../UI/Button/Button";  import "./CourseInput.css";  const CourseInput = (props) => {  const [enteredValue, setEnteredValue] = useState("");  const goalInputChangeHandler = (event) => {  setEnteredValue(event.target.value);  };  const formSubmitHandler = (event) => {  event.preventDefault();  **if (enteredValue.trim().length === 0) {**  **}**  props.onAddGoal(enteredValue);  };  return (  <form onSubmit={formSubmitHandler}>  <div className="form-control">  <label>Course Goal</label>  <input type="text" onChange={goalInputChangeHandler} />  </div>  <Button type="submit">Add Goal</Button>  </form>  );  };  export default CourseInput; |

In that case I want to return; hence the line props.onAddGoal(enteredValue) will not be executed because functional execution stops when you return.

|  |
| --- |
| src/components/CourseGoals/CourseInput/CourseInput.js |
| import React, { useState } from "react";  import Button from "../../UI/Button/Button";  import "./CourseInput.css";  const CourseInput = (props) => {  const [enteredValue, setEnteredValue] = useState("");  const goalInputChangeHandler = (event) => {  setEnteredValue(event.target.value);  };  const formSubmitHandler = (event) => {  event.preventDefault();  if (enteredValue.trim().length === 0) {  **return;**  }  props.onAddGoal(enteredValue); // line will not be executed if the if statement evaluates to true  };  return (  <form onSubmit={formSubmitHandler}>  <div className="form-control">  <label>Course Goal</label>  <input type="text" onChange={goalInputChangeHandler} />  </div>  <Button type="submit">Add Goal</Button>  </form>  );  };  export default CourseInput; |

We made sure that we cannot add empty goals, but the problem now is we don't give the user any feedback. That's exactly where we now need our styling. I want to add a red border and a slightly red background color to that input and also a red color to that label if the user entered something invalid. Now how can we achieve this? Well, we can manage an extra piece of state here, and that State can be an indicator as to whether the user submitted and entered something valid or not. So, it's a true or false choice, and therefore, we'll be working with a boolean. Initially, let's say we trust the user, so we set useState(true)

|  |
| --- |
| src/components/CourseGoals/CourseInput/CourseInput.js |
| import React, { useState } from "react";  import Button from "../../UI/Button/Button";  import "./CourseInput.css";  const CourseInput = (props) => {  const [enteredValue, setEnteredValue] = useState("");  **useState(true)**  const goalInputChangeHandler = (event) => {  setEnteredValue(event.target.value);  };  const formSubmitHandler = (event) => {  event.preventDefault();  if (enteredValue.trim().length === 0) {  return;  }  props.onAddGoal(enteredValue);  };  return (  <form onSubmit={formSubmitHandler}>  <div className="form-control">  <label>Course Goal</label>  <input type="text" onChange={goalInputChangeHandler} />  </div>  <Button type="submit">Add Goal</Button>  </form>  );  };  export default CourseInput; |

And I'll name this state isValid and the updating function setIsValid. As long as this is true, I consider the input as valid.

|  |
| --- |
| src/components/CourseGoals/CourseInput/CourseInput.js |
| import React, { useState } from "react";  import Button from "../../UI/Button/Button";  import "./CourseInput.css";  const CourseInput = (props) => {  const [enteredValue, setEnteredValue] = useState("");  **const [isValid, setIsValid] = useState(true);**  const goalInputChangeHandler = (event) => {  setEnteredValue(event.target.value);  };  const formSubmitHandler = (event) => {  event.preventDefault();  if (enteredValue.trim().length === 0) {  return;  }  props.onAddGoal(enteredValue);  };  return (  <form onSubmit={formSubmitHandler}>  <div className="form-control">  <label>Course Goal</label>  <input type="text" onChange={goalInputChangeHandler} />  </div>  <Button type="submit">Add Goal</Button>  </form>  );  };  export default CourseInput; |

If we make it into this if block,

|  |
| --- |
| src/components/CourseGoals/CourseInput/CourseInput.js |
| import React, { useState } from "react";  import Button from "../../UI/Button/Button";  import "./CourseInput.css";  const CourseInput = (props) => {  const [enteredValue, setEnteredValue] = useState("");  const [isValid, setIsValid] = useState(true);  const goalInputChangeHandler = (event) => {  setEnteredValue(event.target.value);  };  const formSubmitHandler = (event) => {  event.preventDefault();  if (enteredValue.trim().length === 0) {  return;  }  props.onAddGoal(enteredValue);  };  return (  <form onSubmit={formSubmitHandler}>  <div className="form-control">  <label>Course Goal</label>  <input type="text" onChange={goalInputChangeHandler} />  </div>  <Button type="submit">Add Goal</Button>  </form>  );  };  export default CourseInput; |

we know it's invalid and therefore we can call setIsValid(false); which sets isValid to false because what the user entered and submitted definitely is not valid. Now if isValid is false, we want to apply the styles we mentioned.

|  |
| --- |
| src/components/CourseGoals/CourseInput/CourseInput.js |
| import React, { useState } from "react";  import Button from "../../UI/Button/Button";  import "./CourseInput.css";  const CourseInput = (props) => {  const [enteredValue, setEnteredValue] = useState("");  const [isValid, setIsValid] = useState(true);  const goalInputChangeHandler = (event) => {  setEnteredValue(event.target.value);  };  const formSubmitHandler = (event) => {  event.preventDefault();  if (enteredValue.trim().length === 0) {  **setIsValid(false);**  return;  }  props.onAddGoal(enteredValue);  };  return (  <form onSubmit={formSubmitHandler}>  <div className="form-control">  <label>Course Goal</label>  <input type="text" onChange={goalInputChangeHandler} />  </div>  <Button type="submit">Add Goal</Button>  </form>  );  };  export default CourseInput; |

The easiest way of doing that is adding the inline style here to the label for example and as you learned you need to set an object, the inline style prop wants an object as a value. And in that object, you target different CSS style props in JavaScript that you can set for this element for this component.

|  |
| --- |
| src/components/CourseGoals/CourseInput/CourseInput.js |
| import React, { useState } from "react";  import Button from "../../UI/Button/Button";  import "./CourseInput.css";  const CourseInput = (props) => {  const [enteredValue, setEnteredValue] = useState("");  const [isValid, setIsValid] = useState(true);  const goalInputChangeHandler = (event) => {  setEnteredValue(event.target.value);  };  const formSubmitHandler = (event) => {  event.preventDefault();  if (enteredValue.trim().length === 0) {  setIsValid(false);  return;  }  props.onAddGoal(enteredValue);  };  return (  <form onSubmit={formSubmitHandler}>  <div className="form-control">  <label **style={{}}**>Course Goal</label>  <input type="text" onChange={goalInputChangeHandler} />  </div>  <Button type="submit">Add Goal</Button>  </form>  );  };  export default CourseInput; |

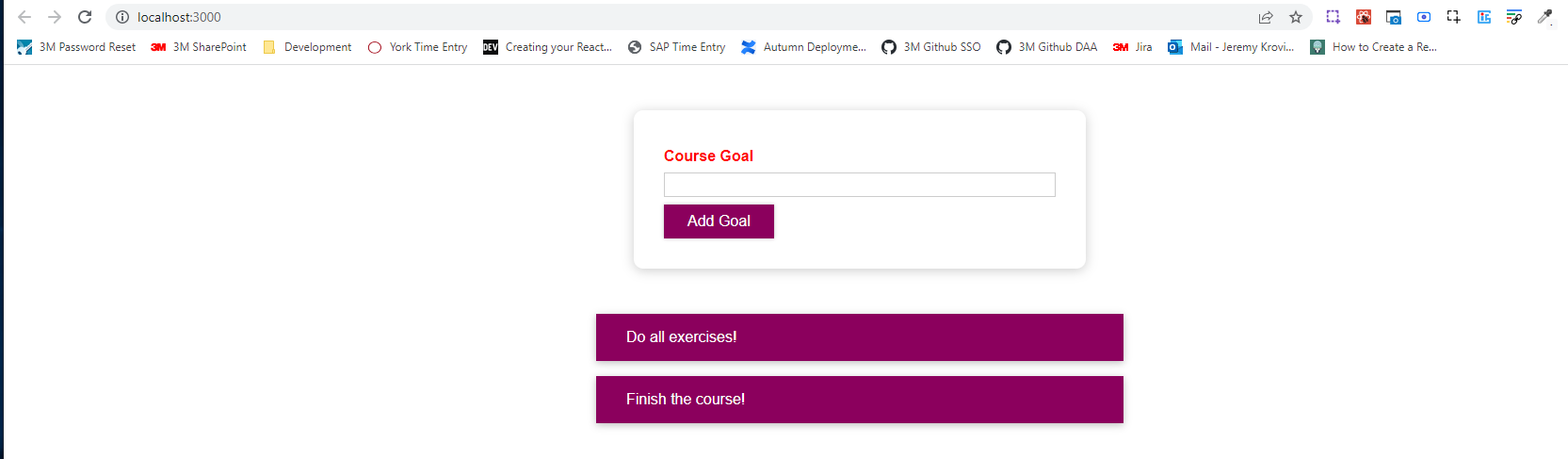
So, for example, here the color property is available and we could set this to "red" but of course we don't want to always set this to "red" but only if the input is invalid.

|  |
| --- |
| src/components/CourseGoals/CourseInput/CourseInput.js |
| import React, { useState } from "react";  import Button from "../../UI/Button/Button";  import "./CourseInput.css";  const CourseInput = (props) => {  const [enteredValue, setEnteredValue] = useState("");  const [isValid, setIsValid] = useState(true);  const goalInputChangeHandler = (event) => {  setEnteredValue(event.target.value);  };  const formSubmitHandler = (event) => {  event.preventDefault();  if (enteredValue.trim().length === 0) {  setIsValid(false);  return;  }  props.onAddGoal(enteredValue);  };  return (  <form onSubmit={formSubmitHandler}>  <div className="form-control">  <label style={{ **color: "red"** }}>Course Goal</label>  <input type="text" onChange={goalInputChangeHandler} />  </div>  <Button type="submit">Add Goal</Button>  </form>  );  };  export default CourseInput; |

So, I check if !isValid, and if that is the case, I set the color to "red", otherwise, we set the color to "black" for example, which is the default color.

|  |
| --- |
| src/components/CourseGoals/CourseInput/CourseInput.js |
| import React, { useState } from "react";  import Button from "../../UI/Button/Button";  import "./CourseInput.css";  const CourseInput = (props) => {  const [enteredValue, setEnteredValue] = useState("");  const [isValid, setIsValid] = useState(true);  const goalInputChangeHandler = (event) => {  setEnteredValue(event.target.value);  };  const formSubmitHandler = (event) => {  event.preventDefault();  if (enteredValue.trim().length === 0) {  setIsValid(false);  return;  }  props.onAddGoal(enteredValue);  };  return (  <form onSubmit={formSubmitHandler}>  <div className="form-control">  <label style={{ **color: !isValid ? "red" : "black"** }}>Course Goal</label>  <input type="text" onChange={goalInputChangeHandler} />  </div>  <Button type="submit">Add Goal</Button>  </form>  );  };  export default CourseInput; |

And that should allow us to dynamically set the label to the color red. We use the style prop, passing an object to it as demanded, but we dynamically changed the value that's fed into the color property in that style object. And, therefore, now if we saved this and we go back, if we try to submit the form without entering anything, the label becomes red.



We also need to change the border and background color of this input.

|  |
| --- |
| src/components/CourseGoals/CourseInput/CourseInput.js |
| import React, { useState } from "react";  import Button from "../../UI/Button/Button";  import "./CourseInput.css";  const CourseInput = (props) => {  const [enteredValue, setEnteredValue] = useState("");  const [isValid, setIsValid] = useState(true);  const goalInputChangeHandler = (event) => {  setEnteredValue(event.target.value);  };  const formSubmitHandler = (event) => {  event.preventDefault();  if (enteredValue.trim().length === 0) {  setIsValid(false);  return;  }  props.onAddGoal(enteredValue);  };  return (  <form onSubmit={formSubmitHandler}>  <div className="form-control">  <label style={{ color: !isValid ? "red" : "black" }}>Course Goal</label>  **<input type="text" onChange={goalInputChangeHandler} />**  </div>  <Button type="submit">Add Goal</Button>  </form>  );  };  export default CourseInput; |

My attempt at changing the border and background color of this input using dynamic inline style:

|  |
| --- |
| src/components/CourseGoals/CourseInput/CourseInput.js |
| import React, { useState } from "react";  import Button from "../../UI/Button/Button";  import "./CourseInput.css";  const CourseInput = (props) => {  const [enteredValue, setEnteredValue] = useState("");  const [isValid, setIsValid] = useState(true);  const goalInputChangeHandler = (event) => {  setEnteredValue(event.target.value);  };  const formSubmitHandler = (event) => {  event.preventDefault();  if (enteredValue.trim().length === 0) {  setIsValid(false);  return;  }  props.onAddGoal(enteredValue);  };  return (  <form onSubmit={formSubmitHandler}>  <div className="form-control">  <label style={{ color: !isValid ? "red" : "black" }}>Course Goal</label>  <input  type="text"  style={{  **border: !isValid ? "1px solid red" : "1px solid black",**  **backgroundColor: !isValid ? "red" : "white",**  }}  onChange={goalInputChangeHandler}  />  </div>  <Button type="submit">Add Goal</Button>  </form>  );  };  export default CourseInput; |

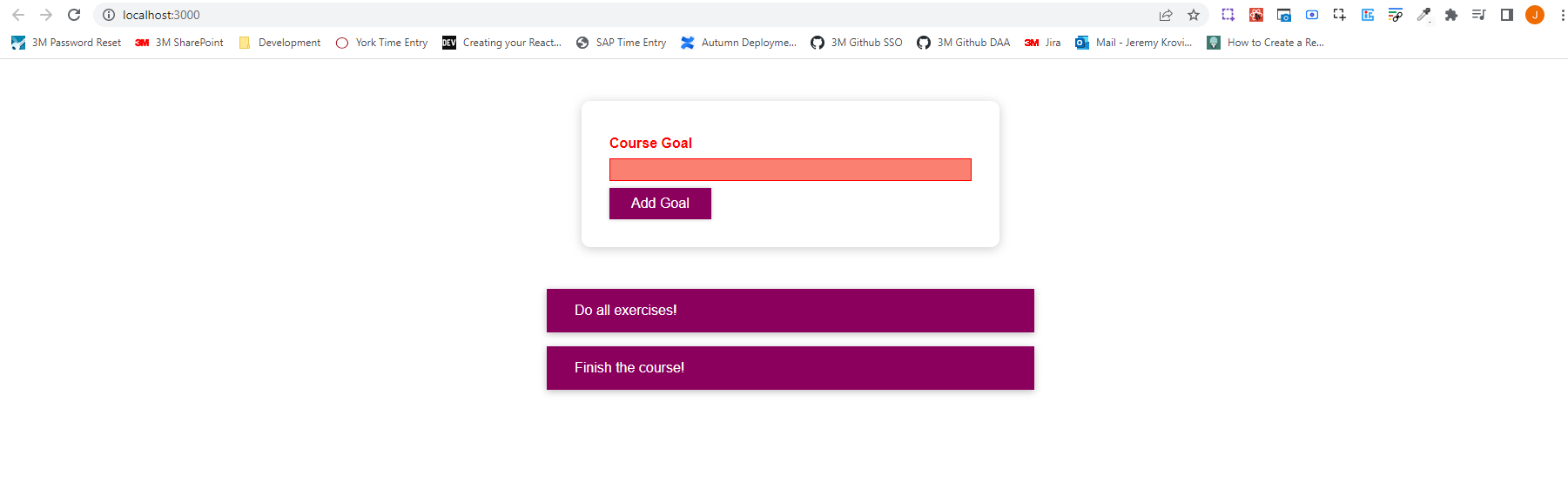
In Max's solution, we need to set the BorderColor to "red" and do what we did for the label. We could also refactor !isValid ? 'red' : 'black' into a separate variable, but we won't because we only need this in these two places.

|  |
| --- |
| src/components/CourseGoals/CourseInput/CourseInput.js |
| import React, { useState } from "react";  import Button from "../../UI/Button/Button";  import "./CourseInput.css";  const CourseInput = (props) => {  const [enteredValue, setEnteredValue] = useState("");  const [isValid, setIsValid] = useState(true);  const goalInputChangeHandler = (event) => {  setEnteredValue(event.target.value);  };  const formSubmitHandler = (event) => {  event.preventDefault();  if (enteredValue.trim().length === 0) {  setIsValid(false);  return;  }  props.onAddGoal(enteredValue);  };  return (  <form onSubmit={formSubmitHandler}>  <div className="form-control">  <label style={{ color: !isValid ? "red" : "black" }}>Course Goal</label>  <input  type="text"  style={{  **borderColor: !isValid ? "red" : "black",**  }}  onChange={goalInputChangeHandler}  />  </div>  <Button type="submit">Add Goal</Button>  </form>  );  };  export default CourseInput; |

In Max's solution he says that we also need a background where we do the same check !isValid ? 'red' : black. This 'red' might be a bit too much and, therefore, we might want to use something else, such as 'salmon' and instead of 'black' in the other case, we will have 'transparent' because of course we don't want a black background in the input.

|  |
| --- |
| src/components/CourseGoals/CourseInput/CourseInput.js |
| import React, { useState } from "react";  import Button from "../../UI/Button/Button";  import "./CourseInput.css";  const CourseInput = (props) => {  const [enteredValue, setEnteredValue] = useState("");  const [isValid, setIsValid] = useState(true);  const goalInputChangeHandler = (event) => {  setEnteredValue(event.target.value);  };  const formSubmitHandler = (event) => {  event.preventDefault();  if (enteredValue.trim().length === 0) {  setIsValid(false);  return;  }  props.onAddGoal(enteredValue);  };  return (  <form onSubmit={formSubmitHandler}>  <div className="form-control">  <label style={{ color: !isValid ? "red" : "black" }}>Course Goal</label>  <input  type="text"  style={{  borderColor: !isValid ? "red" : "black",  **background: !isValid ? "salmon" : "transparent",**  }}  onChange={goalInputChangeHandler}  />  </div>  <Button type="submit">Add Goal</Button>  </form>  );  };  export default CourseInput; |

When we save this, if we reload the browser and click "Add Goal" without any input, we get the salmon background and red border as shown below.



At the moment, the styling is never reset, so if we don't input anything at first and click "Add Goal", but then we actually input something, the background color will still be salmon and the border will still be red.

Two notes: for one of course we want to reset the background eventually, and second, I’m not entirely happy with this approach we currently have because we always set inline styles, which of course have the highest priority in CSS, so you will override all other styles with that.

I am happy with the styles set up in CourseInput.css where I have a border with a light greyish color, #ccc. We could set this up in our inline styles where we set borderColor: !isValid ? 'red' : '#ccc' instead of black for example, so that when we reload the page, we go back to this predefined color we have in the CSS file.

|  |
| --- |
| src/components/CourseGoals/CourseInput/CourseInput.js |
| import React, { useState } from "react";  import Button from "../../UI/Button/Button";  import "./CourseInput.css";  const CourseInput = (props) => {  const [enteredValue, setEnteredValue] = useState("");  const [isValid, setIsValid] = useState(true);  const goalInputChangeHandler = (event) => {  setEnteredValue(event.target.value);  };  const formSubmitHandler = (event) => {  event.preventDefault();  if (enteredValue.trim().length === 0) {  setIsValid(false);  return;  }  props.onAddGoal(enteredValue);  };  return (  <form onSubmit={formSubmitHandler}>  <div className="form-control">  <label style={{ color: !isValid ? "red" : "black" }}>Course Goal</label>  <input  type="text"  style={{  borderColor: !isValid ? "red" : "#**ccc**",  background: !isValid ? "salmon" : "transparent",  }}  onChange={goalInputChangeHandler}  />  </div>  <Button type="submit">Add Goal</Button>  </form>  );  };  export default CourseInput; |

We still have some duplication. We have a base color scheme here in the CSS file.

|  |
| --- |
| src/components/CourseGoals/CourseInput/CourseInput.css |
| .form-control {  margin: 0.5rem 0;  }  .form-control label {  font-weight: bold;  display: block;  margin-bottom: 0.5rem;  }  .form-control input {  display: block;  width: 100%;  border: 1px solid **#ccc**;  font: inherit;  line-height: 1.5rem;  padding: 0 0.25rem;  }  .form-control input:focus {  outline: none;  background: #fad0ec;  border-color: #8b005d;  } |

And then we override this with inline styles just we want to conditionally set some specific styles if the input is invalid.

|  |
| --- |
| src/components/CourseGoals/CourseInput/CourseInput.js |
| import React, { useState } from "react";  import Button from "../../UI/Button/Button";  import "./CourseInput.css";  const CourseInput = (props) => {  const [enteredValue, setEnteredValue] = useState("");  const [isValid, setIsValid] = useState(true);  const goalInputChangeHandler = (event) => {  setEnteredValue(event.target.value);  };  const formSubmitHandler = (event) => {  event.preventDefault();  if (enteredValue.trim().length === 0) {  setIsValid(false);  return;  }  props.onAddGoal(enteredValue);  };  return (  <form onSubmit={formSubmitHandler}>  <div className="form-control">  <label style={{ color: !isValid ? "red" : "black" }}>Course Goal</label>  <input  type="text"  style={{  borderColor: !isValid ? "red" : "#**ccc**",  background: !isValid ? "salmon" : "transparent",  }}  onChange={goalInputChangeHandler}  />  </div>  <Button type="submit">Add Goal</Button>  </form>  );  };  export default CourseInput; |

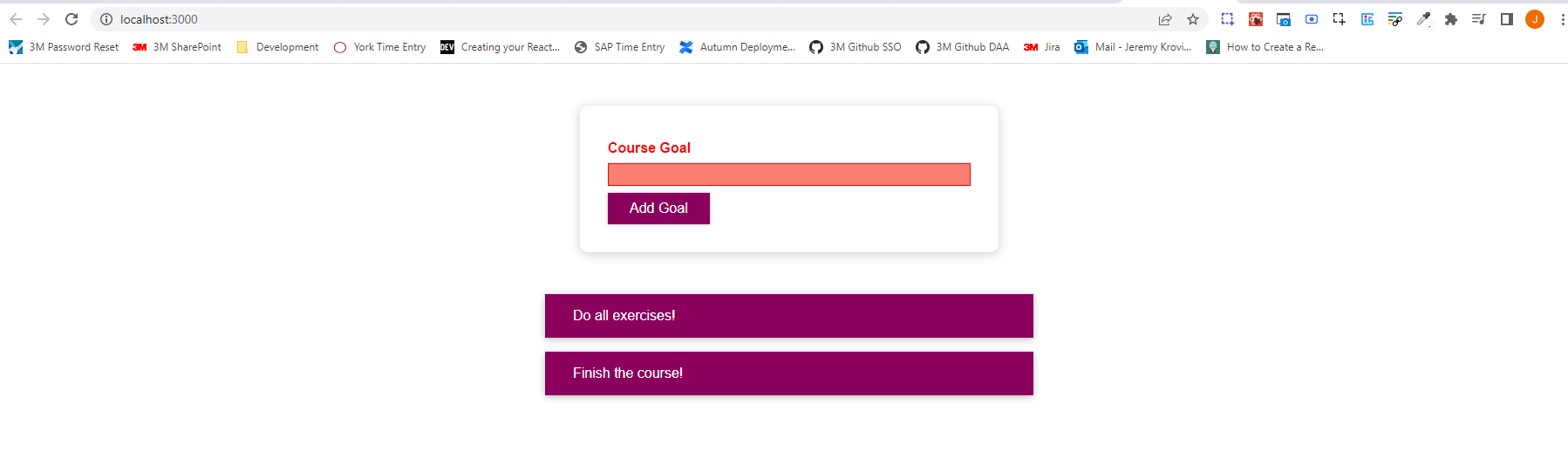
We will now quickly work on the reset functionality before we are shown an alternative to setting dynamic styles with inline styles. For the reset functionality, all we need to do is go to the place where we react to every keystroke and in the end we just need to check if event.target.value (event.target.value is the entered text) is now basically valid, so if we trim it and get the length and check if the length is greater than 0.

|  |
| --- |
| src/components/CourseGoals/CourseInput/CourseInput.js |
| import React, { useState } from "react";  import Button from "../../UI/Button/Button";  import "./CourseInput.css";  const CourseInput = (props) => {  const [enteredValue, setEnteredValue] = useState("");  const [isValid, setIsValid] = useState(true);  const goalInputChangeHandler = (event) => {  **if (event.target.value.trim().length > 0) {**  **}**  setEnteredValue(event.target.value);  };  const formSubmitHandler = (event) => {  event.preventDefault();  if (enteredValue.trim().length === 0) {  setIsValid(false);  return;  }  props.onAddGoal(enteredValue);  };  return (  <form onSubmit={formSubmitHandler}>  <div className="form-control">  <label style={{ color: !isValid ? "red" : "black" }}>Course Goal</label>  <input  type="text"  style={{  borderColor: !isValid ? "red" : "#ccc",  background: !isValid ? "salmon" : "transparent",  }}  onChange={goalInputChangeHandler}  />  </div>  <Button type="submit">Add Goal</Button>  </form>  );  };  export default CourseInput; |

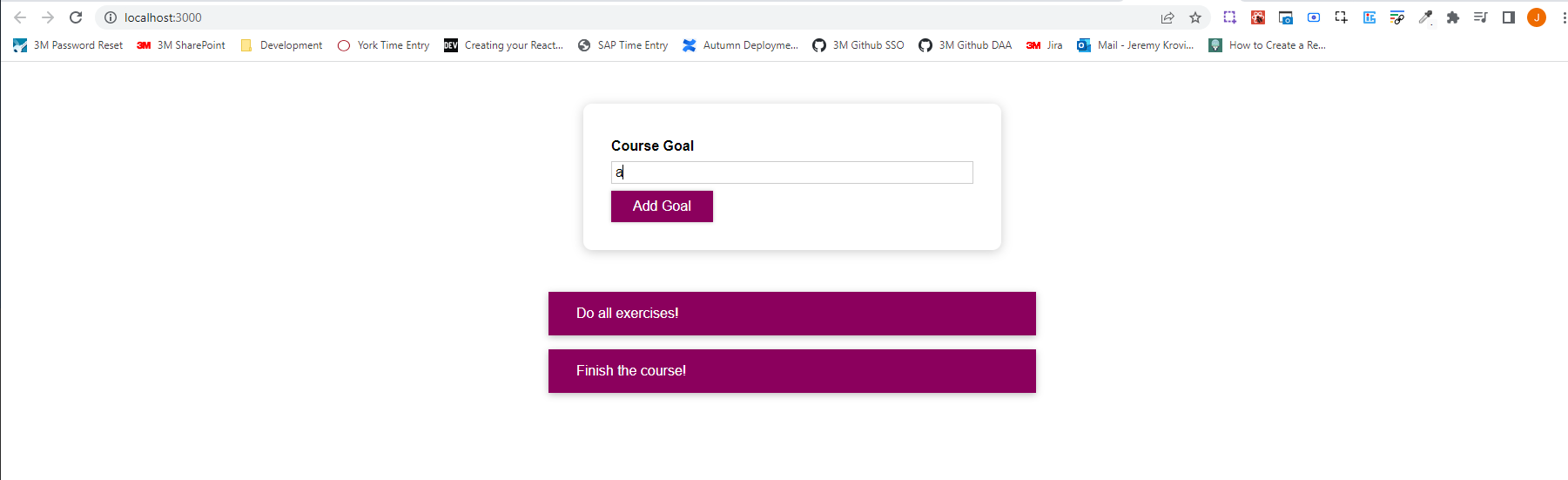
If we check that the length of the trimmed input is greater than zero, we can call setIsValid(true), which sets isValid to true again.

|  |
| --- |
| src/components/CourseGoals/CourseInput/CourseInput.js |
| import React, { useState } from "react";  import Button from "../../UI/Button/Button";  import "./CourseInput.css";  const CourseInput = (props) => {  const [enteredValue, setEnteredValue] = useState("");  const [isValid, setIsValid] = useState(true);  const goalInputChangeHandler = (event) => {  if (event.target.value.trim().length > 0) {  **setIsValid(true);**  }  setEnteredValue(event.target.value);  };  const formSubmitHandler = (event) => {  event.preventDefault();  if (enteredValue.trim().length === 0) {  setIsValid(false);  return;  }  props.onAddGoal(enteredValue);  };  return (  <form onSubmit={formSubmitHandler}>  <div className="form-control">  <label style={{ color: !isValid ? "red" : "black" }}>Course Goal</label>  <input  type="text"  style={{  borderColor: !isValid ? "red" : "#ccc",  background: !isValid ? "salmon" : "transparent",  }}  onChange={goalInputChangeHandler}  />  </div>  <Button type="submit">Add Goal</Button>  </form>  );  };  export default CourseInput; |

If we save and reload the browser, when we submit without any input, we see the red label and border as well as the salmon background



but if we start typing, you'll see that the input no longer shows as being invalid.



### 76. Setting CSS Classes Dynamically

What if we could add a class to this div which holds both the label and the input,

|  |
| --- |
| src/components/CourseGoals/CourseInput/CourseInput.js |
| import React, { useState } from "react";  import Button from "../../UI/Button/Button";  import "./CourseInput.css";  const CourseInput = (props) => {  const [enteredValue, setEnteredValue] = useState("");  const [isValid, setIsValid] = useState(true);  const goalInputChangeHandler = (event) => {  if (event.target.value.trim().length > 0) {  setIsValid(true);  }  setEnteredValue(event.target.value);  };  const formSubmitHandler = (event) => {  event.preventDefault();  if (enteredValue.trim().length === 0) {  setIsValid(false);  return;  }  props.onAddGoal(enteredValue);  };  return (  <form onSubmit={formSubmitHandler}>  **<div className="form-control">**  <label style={{ color: !isValid ? "red" : "black" }}>Course Goal</label>  <input  type="text"  style={{  borderColor: !isValid ? "red" : "#ccc",  background: !isValid ? "salmon" : "transparent",  }}  onChange={goalInputChangeHandler}  />  </div>  <Button type="submit">Add Goal</Button>  </form>  );  };  export default CourseInput; |

And we would add this class dynamically only if the value entered is invalid. It could be something like an invalid class here. The name is totally up to you. So this invalid class should sometimes be added here on this div and then in CSS we could prepare the fitting styles.

|  |
| --- |
| src/components/CourseGoals/CourseInput/CourseInput.js |
| import React, { useState } from "react";  import Button from "../../UI/Button/Button";  import "./CourseInput.css";  const CourseInput = (props) => {  const [enteredValue, setEnteredValue] = useState("");  const [isValid, setIsValid] = useState(true);  const goalInputChangeHandler = (event) => {  if (event.target.value.trim().length > 0) {  setIsValid(true);  }  setEnteredValue(event.target.value);  };  const formSubmitHandler = (event) => {  event.preventDefault();  if (enteredValue.trim().length === 0) {  setIsValid(false);  return;  }  props.onAddGoal(enteredValue);  };  return (  <form onSubmit={formSubmitHandler}>  <div className="form-control **invalid**">  <label style={{ color: !isValid ? "red" : "black" }}>Course Goal</label>  <input  type="text"  style={{  borderColor: !isValid ? "red" : "#ccc",  background: !isValid ? "salmon" : "transparent",  }}  onChange={goalInputChangeHandler}  />  </div>  <Button type="submit">Add Goal</Button>  </form>  );  };  export default CourseInput; |

For example, at the bottom of this file, we can add some style where we say if the form-control also has the invalid class, which we represent as .form-control.invalid in CSS (make sure there is no space between .form-control, the ., and invalid, which means they both have to be on the same element). Then we could target our input and set border-color: red, and we could set background: #ffd7d7;

|  |
| --- |
| src/components/CourseGoals/CourseInput/CourseInput.css |
| .form-control {  margin: 0.5rem 0;  }  .form-control label {  font-weight: bold;  display: block;  margin-bottom: 0.5rem;  }  .form-control input {  display: block;  width: 100%;  border: 1px solid #ccc;  font: inherit;  line-height: 1.5rem;  padding: 0 0.25rem;  }  .form-control input:focus {  outline: none;  background: #fad0ec;  border-color: #8b005d;  }  **.form-control.invalid input {**  **border-color: red;**  **background: #ffd7d7;**  **}** |

And, if the form-control, the div with the class form-control has a class of invlaid, we can also target the label and set its color to red.

|  |
| --- |
| src/components/CourseGoals/CourseInput/CourseInput.css |
| .form-control {  margin: 0.5rem 0;  }  .form-control label {  font-weight: bold;  display: block;  margin-bottom: 0.5rem;  }  .form-control input {  display: block;  width: 100%;  border: 1px solid #ccc;  font: inherit;  line-height: 1.5rem;  padding: 0 0.25rem;  }  .form-control input:focus {  outline: none;  background: #fad0ec;  border-color: #8b005d;  }  .form-control.invalid input {  border-color: red;  background: #ffd7d7;  }  **.form-control.invalid label {**  **color: red;**  **}** |

So these here are two CSS rules, and these rules should lead to the desired output. All we need to do for these rules to have an effect is we need to add that invalid class dynamically.

We will start by getting rid of the inline styles.

|  |
| --- |
| src/components/CourseGoals/CourseInput/CourseInput.js |
| import React, { useState } from "react";  import Button from "../../UI/Button/Button";  import "./CourseInput.css";  const CourseInput = (props) => {  const [enteredValue, setEnteredValue] = useState("");  const [isValid, setIsValid] = useState(true);  const goalInputChangeHandler = (event) => {  if (event.target.value.trim().length > 0) {  setIsValid(true);  }  setEnteredValue(event.target.value);  };  const formSubmitHandler = (event) => {  event.preventDefault();  if (enteredValue.trim().length === 0) {  setIsValid(false);  return;  }  props.onAddGoal(enteredValue);  };  return (  <form onSubmit={formSubmitHandler}>  <div className="form-control ">  <label ~~style={{ color: !isValid ? "red" : "black" }}~~>Course Goal</label>  <input  type="text"  ~~style={{~~  ~~borderColor: !isValid ? "red" : "#ccc",~~  ~~background: !isValid ? "salmon" : "transparent",~~  ~~}}~~  onChange={goalInputChangeHandler}  />  </div>  <Button type="submit">Add Goal</Button>  </form>  );  };  export default CourseInput; |

The component file without the inline styles:

|  |
| --- |
| src/components/CourseGoals/CourseInput/CourseInput.js |
| import React, { useState } from "react";  import Button from "../../UI/Button/Button";  import "./CourseInput.css";  const CourseInput = (props) => {  const [enteredValue, setEnteredValue] = useState("");  const [isValid, setIsValid] = useState(true);  const goalInputChangeHandler = (event) => {  if (event.target.value.trim().length > 0) {  setIsValid(true);  }  setEnteredValue(event.target.value);  };  const formSubmitHandler = (event) => {  event.preventDefault();  if (enteredValue.trim().length === 0) {  setIsValid(false);  return;  }  props.onAddGoal(enteredValue);  };  return (  <form onSubmit={formSubmitHandler}>  <div className="form-control">  <label>Course Goal</label>  <input type="text" onChange={goalInputChangeHandler} />  </div>  <Button type="submit">Add Goal</Button>  </form>  );  };  export default CourseInput; |

Since we need a dynamic value, we need curly braces.

|  |
| --- |
| src/components/CourseGoals/CourseInput/CourseInput.js |
| import React, { useState } from "react";  import Button from "../../UI/Button/Button";  import "./CourseInput.css";  const CourseInput = (props) => {  const [enteredValue, setEnteredValue] = useState("");  const [isValid, setIsValid] = useState(true);  const goalInputChangeHandler = (event) => {  if (event.target.value.trim().length > 0) {  setIsValid(true);  }  setEnteredValue(event.target.value);  };  const formSubmitHandler = (event) => {  event.preventDefault();  if (enteredValue.trim().length === 0) {  setIsValid(false);  return;  }  props.onAddGoal(enteredValue);  };  return (  <form onSubmit={formSubmitHandler}>  <div className=**{}**>  <label>Course Goal</label>  <input type="text" onChange={goalInputChangeHandler} />  </div>  <Button type="submit">Add Goal</Button>  </form>  );  };  export default CourseInput; |

Now what we feed into the curly braces still can be a string of "form-control", but a string where we dynamically add more text in it, where we dynamically expand it or

|  |
| --- |
| src/components/CourseGoals/CourseInput/CourseInput.js |
| import React, { useState } from "react";  import Button from "../../UI/Button/Button";  import "./CourseInput.css";  const CourseInput = (props) => {  const [enteredValue, setEnteredValue] = useState("");  const [isValid, setIsValid] = useState(true);  const goalInputChangeHandler = (event) => {  if (event.target.value.trim().length > 0) {  setIsValid(true);  }  setEnteredValue(event.target.value);  };  const formSubmitHandler = (event) => {  event.preventDefault();  if (enteredValue.trim().length === 0) {  setIsValid(false);  return;  }  props.onAddGoal(enteredValue);  };  return (  <form onSubmit={formSubmitHandler}>  <div className={**"form-control"**}>  <label>Course Goal</label>  <input type="text" onChange={goalInputChangeHandler} />  </div>  <Button type="submit">Add Goal</Button>  </form>  );  };  export default CourseInput; |

concatenate a value to have this string "form-control invalid"

|  |
| --- |
| src/components/CourseGoals/CourseInput/CourseInput.js |
| import React, { useState } from "react";  import Button from "../../UI/Button/Button";  import "./CourseInput.css";  const CourseInput = (props) => {  const [enteredValue, setEnteredValue] = useState("");  const [isValid, setIsValid] = useState(true);  const goalInputChangeHandler = (event) => {  if (event.target.value.trim().length > 0) {  setIsValid(true);  }  setEnteredValue(event.target.value);  };  const formSubmitHandler = (event) => {  event.preventDefault();  if (enteredValue.trim().length === 0) {  setIsValid(false);  return;  }  props.onAddGoal(enteredValue);  };  return (  <form onSubmit={formSubmitHandler}>  <div className={**"form-control invalid"**}>  <label>Course Goal</label>  <input type="text" onChange={goalInputChangeHandler} />  </div>  <Button type="submit">Add Goal</Button>  </form>  );  };  export default CourseInput; |

instead of the classname "form-control"

|  |
| --- |
| src/components/CourseGoals/CourseInput/CourseInput.js |
| import React, { useState } from "react";  import Button from "../../UI/Button/Button";  import "./CourseInput.css";  const CourseInput = (props) => {  const [enteredValue, setEnteredValue] = useState("");  const [isValid, setIsValid] = useState(true);  const goalInputChangeHandler = (event) => {  if (event.target.value.trim().length > 0) {  setIsValid(true);  }  setEnteredValue(event.target.value);  };  const formSubmitHandler = (event) => {  event.preventDefault();  if (enteredValue.trim().length === 0) {  setIsValid(false);  return;  }  props.onAddGoal(enteredValue);  };  return (  <form onSubmit={formSubmitHandler}>  <div className={**"form-control"**}>  <label>Course Goal</label>  <input type="text" onChange={goalInputChangeHandler} />  </div>  <Button type="submit">Add Goal</Button>  </form>  );  };  export default CourseInput; |

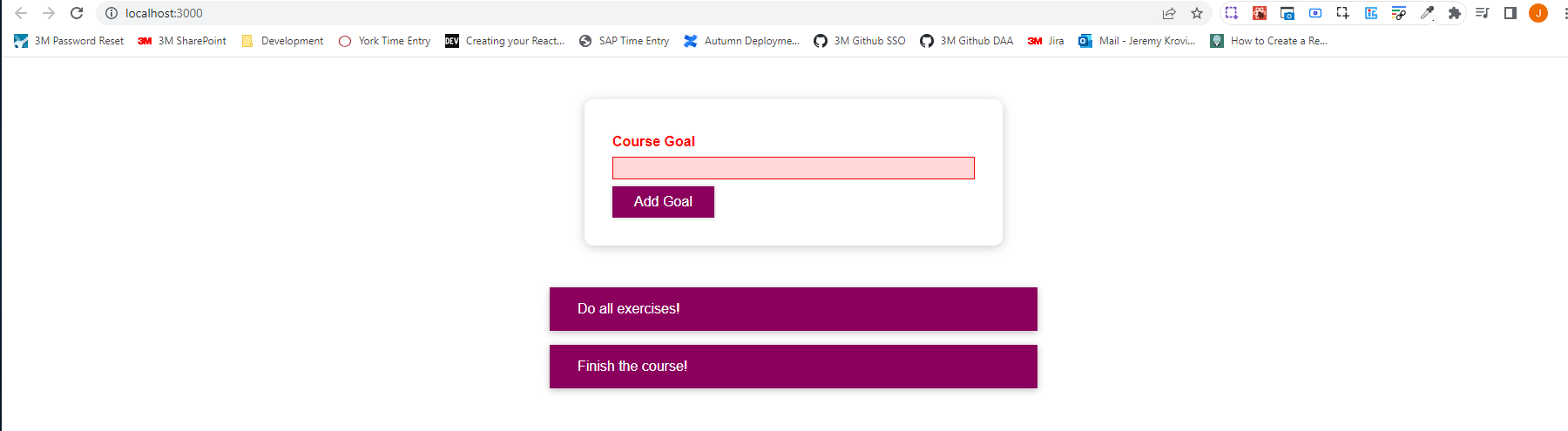
So what we pass to className for this div will sometimes be a string that looks like this "form-control" and other times will be a string that looks like this "form-control invalid" depending on what the user entered and depending on our isValid state. We can use a nice construct. We can use backticks, a template literal in JavaScript so we can add `form-control`,

|  |
| --- |
| src/components/CourseGoals/CourseInput/CourseInput.js |
| import React, { useState } from "react";  import Button from "../../UI/Button/Button";  import "./CourseInput.css";  const CourseInput = (props) => {  const [enteredValue, setEnteredValue] = useState("");  const [isValid, setIsValid] = useState(true);  const goalInputChangeHandler = (event) => {  if (event.target.value.trim().length > 0) {  setIsValid(true);  }  setEnteredValue(event.target.value);  };  const formSubmitHandler = (event) => {  event.preventDefault();  if (enteredValue.trim().length === 0) {  setIsValid(false);  return;  }  props.onAddGoal(enteredValue);  };  return (  <form onSubmit={formSubmitHandler}>  <div className={**`form-control`**}>  <label>Course Goal</label>  <input type="text" onChange={goalInputChangeHandler} />  </div>  <Button type="submit">Add Goal</Button>  </form>  );  };  export default CourseInput; |

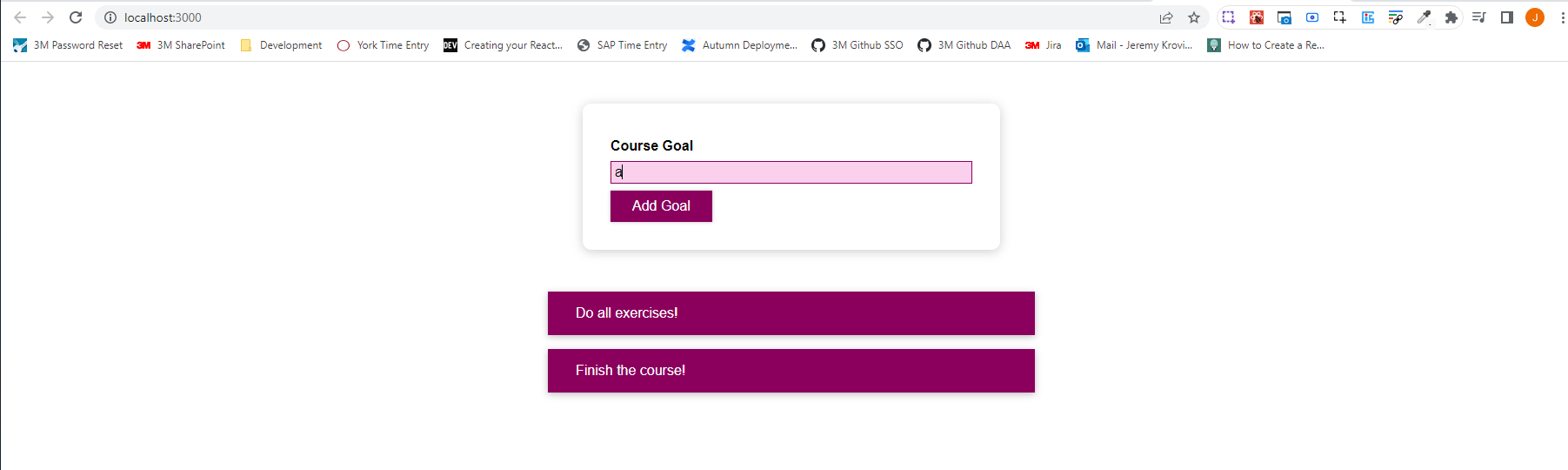
you can also inject a dynamic value into that string with a special syntax that looks like this: ${}. A dollar sign and the a pair of curly braces adds content into the string and what you pass between these curly braces can be any JavaScript expression, so a lot like our curly braces inside of JSX. So now we can check if isValid is not true and if that's the case we insert 'invalid' otherwise we enter nothing or have an empty string in this case: ${!isValid ? 'invalid' : ''}, so this will make sure that className is either set to a string which only has "formControl " or to a string which has "form-conrtrol invalid".

|  |
| --- |
| src/components/CourseGoals/CourseInput/CourseInput.js |
| import React, { useState } from "react";  import Button from "../../UI/Button/Button";  import "./CourseInput.css";  const CourseInput = (props) => {  const [enteredValue, setEnteredValue] = useState("");  const [isValid, setIsValid] = useState(true);  const goalInputChangeHandler = (event) => {  if (event.target.value.trim().length > 0) {  setIsValid(true);  }  setEnteredValue(event.target.value);  };  const formSubmitHandler = (event) => {  event.preventDefault();  if (enteredValue.trim().length === 0) {  setIsValid(false);  return;  }  props.onAddGoal(enteredValue);  };  return (  <form onSubmit={formSubmitHandler}>  <div className={`form-control **${!isValid ? "invalid" : ""}**`}>  <label>Course Goal</label>  <input type="text" onChange={goalInputChangeHandler} />  </div>  <Button type="submit">Add Goal</Button>  </form>  );  };  export default CourseInput; |

Therefore, now we save and reload the page. If we hit "Add Goal" with nothing typed in the input, we get our error style.



Other than that, if we start typing, we go back to the non-error style because we dynamically added the invalid class.



You could of course inject multiple things into that string, so you could repeat the same technique to add more dynamically added classes.

|  |
| --- |
| src/components/CourseGoals/CourseInput/CourseInput.js |
| import React, { useState } from "react";  import Button from "../../UI/Button/Button";  import "./CourseInput.css";  const CourseInput = (props) => {  const [enteredValue, setEnteredValue] = useState("");  const [isValid, setIsValid] = useState(true);  const goalInputChangeHandler = (event) => {  if (event.target.value.trim().length > 0) {  setIsValid(true);  }  setEnteredValue(event.target.value);  };  const formSubmitHandler = (event) => {  event.preventDefault();  if (enteredValue.trim().length === 0) {  setIsValid(false);  return;  }  props.onAddGoal(enteredValue);  };  return (  <form onSubmit={formSubmitHandler}>  <div className={`form-control ${!isValid ? "invalid" : ""} **${}**`}>  <label>Course Goal</label>  <input type="text" onChange={goalInputChangeHandler} />  </div>  <Button type="submit">Add Goal</Button>  </form>  );  };  export default CourseInput; |

and that's very powerful because now you're back to working with CSS files and with classes only, and you can dynamically add or remove classes with this simple syntax. And it will be React's job to actually add or remove the classes in the DOM.

|  |
| --- |
| src/components/CourseGoals/CourseInput/CourseInput.js |
| import React, { useState } from "react";  import Button from "../../UI/Button/Button";  import "./CourseInput.css";  const CourseInput = (props) => {  const [enteredValue, setEnteredValue] = useState("");  const [isValid, setIsValid] = useState(true);  const goalInputChangeHandler = (event) => {  if (event.target.value.trim().length > 0) {  setIsValid(true);  }  setEnteredValue(event.target.value);  };  const formSubmitHandler = (event) => {  event.preventDefault();  if (enteredValue.trim().length === 0) {  setIsValid(false);  return;  }  props.onAddGoal(enteredValue);  };  return (  <form onSubmit={formSubmitHandler}>  <div className={**`form-control ${!isValid ? "invalid" : ""}`**}>  <label>Course Goal</label>  <input type="text" onChange={goalInputChangeHandler} />  </div>  <Button type="submit">Add Goal</Button>  </form>  );  };  export default CourseInput; |

You do what you can do best in React. You specify a goal, you specify some alternative states, for example, that the goal sometimes is to have just form-control and sometimes form-control and invalid and React will make sure that the DOM, the real DOM is updated accordingly. This is React in action and how you can add styles dynamically.

### 77. Introducing Styled Components

We're importing "./CourseInput.css" into our component files, but this does not result in those styles being scoped to the CourseInput component only.

|  |
| --- |
| src/components/CourseGoals/CourseInput/CourseInput.js |
| import React, { useState } from "react";  import Button from "../../UI/Button/Button";  **import "./CourseInput.css";**  const CourseInput = (props) => {  const [enteredValue, setEnteredValue] = useState("");  const [isValid, setIsValid] = useState(true);  const goalInputChangeHandler = (event) => {  if (event.target.value.trim().length > 0) {  setIsValid(true);  }  setEnteredValue(event.target.value);  };  const formSubmitHandler = (event) => {  event.preventDefault();  if (enteredValue.trim().length === 0) {  setIsValid(false);  return;  }  props.onAddGoal(enteredValue);  };  return (  <form onSubmit={formSubmitHandler}>  <div className={`form-control ${!isValid ? "invalid" : ""}`}>  <label>Course Goal</label>  <input type="text" onChange={goalInputChangeHandler} />  </div>  <Button type="submit">Add Goal</Button>  </form>  );  };  export default CourseInput; |

So that means that if we had any other element anywhere else in the DOM that had a form controlled CSS class on it, this style, for example, would target it and would affect it. So, it's not like we only care about form-control elements in this component, no, every element on the page is effected because by default, those styles are not scoped.

|  |
| --- |
| src/components/CourseGoals/CourseInput/CourseInput.css |
| **.form-control {**  **margin: 0.5rem 0;**  **}**  .form-control label {  font-weight: bold;  display: block;  margin-bottom: 0.5rem;  }  .form-control input {  display: block;  width: 100%;  border: 1px solid #ccc;  font: inherit;  line-height: 1.5rem;  padding: 0 0.25rem;  }  .form-control input:focus {  outline: none;  background: #fad0ec;  border-color: #8b005d;  }  .form-control.invalid input {  border-color: red;  background: #ffd7d7;  }  .form-control.invalid label {  color: red;  } |

Not having styles scoped is not necessarily a problem. You can of course be careful about your selectors. You can use class selectors or ID selectors. It is regular CSS after all, you can use any selector, and you can ensure that the IDs and classes you are selecting are only used in the places where you want to apply the styling. But of course, especially in larger projects, potentially with a lot of developers working on the code, it could happen that a name is used twice. That "form-control" class for example could be used in different parts of the application, and therefore, unfortunately styles spill over to other components, and you might want to avoid this.

Approach number 1 to avoid styles spilling over to other components when the same className, ID name, etc. is to use a package called "styled-components." There is a web page for 'styled-components': <https://styled-components.com/>. Styled-components is a package that helps you build components with certain styles attached to them where the styles really only affect the components to which they were attached and not any other components. Now to get started, we need to install that package which you can do with the command, npm install –-save styled-components.

Once you have it installed, we can pick a place where we do want to use it. Let's say we want to style our Button.

|  |
| --- |
| src/components/UI/Button/Button.js |
| import React from 'react';  import './Button.css';  const Button = props => {  return (  <button type={props.type} className="button" onClick={props.onClick}>  {props.children}  </button>  );  };  export default Button; |

We can rebuild our Button by creating a new const Button, but now what we store in there is not a functional component, but instead

|  |
| --- |
| src/components/UI/Button/Button.js |
| import React from 'react';  import './Button.css';  **const** **Button =**  // const Button = props => {  // return (  // <button type={props.type} className="button" onClick={props.onClick}>  // {props.children}  // </button>  // );  // };  export default Button; |

we now import styled from 'styled-components';

|  |
| --- |
| src/components/UI/Button/Button.js |
| import React from 'react';  **import styled from 'styled-components';**  import './Button.css';  const Button =  // const Button = props => {  // return (  // <button type={props.type} className="button" onClick={props.onClick}>  // {props.children}  // </button>  // );  // };  export default Button; |

We can get rid of the "./Button.css" import.

|  |
| --- |
| src/components/UI/Button/Button.js |
| import React from 'react';  **import styled from 'styled-components';**  const Button =  // const Button = props => {  // return (  // <button type={props.type} className="button" onClick={props.onClick}>  // {props.children}  // </button>  // );  // };  export default Button; |

And instead now we use styled.button``; This syntax is called attacked template literal.

|  |
| --- |
| src/components/UI/Button/Button.js |
| import React from "react";  import styled from "styled-components";  const Button = **styled.button``;**  // const Button = props => {  // return (  // <button type={props.type} className="button" onClick={props.onClick}>  // {props.children}  // </button>  // );  // };  export default Button; |

button is simply a method on

|  |
| --- |
| src/components/UI/Button/Button.js |
| import React from "react";  import styled from "styled-components";  const Button = styled.**button``;**  // const Button = props => {  // return (  // <button type={props.type} className="button" onClick={props.onClick}>  // {props.children}  // </button>  // );  // };  export default Button; |

this styled object.

|  |
| --- |
| src/components/UI/Button/Button.js |
| import React from "react";  import styled from "styled-components";  const Button = **styled**.button**``;**  // const Button = props => {  // return (  // <button type={props.type} className="button" onClick={props.onClick}>  // {props.children}  // </button>  // );  // };  export default Button; |

styled is an object that we're importing from 'styled-components', and there we can access the button method. The button method is a special kind of method. It's not a method that you call with parentheses, but instead you add backticks (``) after the method to call it.

In the end, button`` will be executed as a method behind the scenes, and what you pass into the backticks will be passed into this method, just in a special way.

|  |
| --- |
| src/components/UI/Button/Button.js |
| import React from "react";  import styled from "styled-components";  const Button = styled.**button``;**  // const Button = props => {  // return (  // <button type={props.type} className="button" onClick={props.onClick}>  // {props.children}  // </button>  // );  // };  export default Button; |

And, this button`` method will return a new Button component. The 'components-styled' package has methods for all HTML elements. It has a p method for a paragraph, h1, h2, and so on for the h1, h2 tags, div to create a div, and so on.

This overall expression will return a button, but a button with the styles that we pass between

|  |
| --- |
| src/components/UI/Button/Button.js |
| import React from "react";  import styled from "styled-components";  const Button = **styled.button``;**  // const Button = props => {  // return (  // <button type={props.type} className="button" onClick={props.onClick}>  // {props.children}  // </button>  // );  // };  export default Button; |

these two backticks

|  |
| --- |
| src/components/UI/Button/Button.js |
| import React from "react";  import styled from "styled-components";  const Button = styled.button**``**;  // const Button = props => {  // return (  // <button type={props.type} className="button" onClick={props.onClick}>  // {props.children}  // </button>  // );  // };  export default Button; |

With backticks, you can write a multiline string without any extra syntax, like this, for example,

|  |
| --- |
| src/components/UI/Button/Button.js |
| import React from "react";  import styled from "styled-components";  const Button = styled.button`  `;  // const Button = props => {  // return (  // <button type={props.type} className="button" onClick={props.onClick}>  // {props.children}  // </button>  // );  // };  export default Button; |

and now we can just copy all the button styles from "Button.css"

|  |
| --- |
| src/components/UI/Button/Button.css |
| .button {  font: inherit;  padding: 0.5rem 1.5rem;  border: 1px solid #8b005d;  color: white;  background: #8b005d;  box-shadow: 0 0 4px rgba(0, 0, 0, 0.26);  cursor: pointer;  }  .button:focus {  outline: none;  }  .button:hover,  .button:active {  background: #ac0e77;  border-color: #ac0e77;  box-shadow: 0 0 8px rgba(0, 0, 0, 0.26);  } |

And add them in "Button.js"

|  |
| --- |
| src/components/UI/Button/Button.js |
| import React from "react";  import styled from "styled-components";  const Button = styled.button`  **.button {**  **font: inherit;**  **padding: 0.5rem 1.5rem;**  **border: 1px solid #8b005d;**  **color: white;**  **background: #8b005d;**  **box-shadow: 0 0 4px rgba(0, 0, 0, 0.26);**  **cursor: pointer;**  **}**  **.button:focus {**  **outline: none;**  **}**  **.button:hover,**  **.button:active {**  **background: #ac0e77;**  **border-color: #ac0e77;**  **box-shadow: 0 0 8px rgba(0, 0, 0, 0.26);**  **}**  `;  // const Button = props => {  // return (  // <button type={props.type} className="button" onClick={props.onClick}>  // {props.children}  // </button>  // );  // };  export default Button; |

We'll need to tweak them, but that is something we can do. Now what do we need to do to make this work in the end? .button, bolded in the below code snippet, will return a button, and this button will not have a class of button because we have no place where we would set up this class name. So, instead the styles that we pass between the opening and closing backticks will directly affect this button.

|  |
| --- |
| src/components/UI/Button/Button.js |
| import React from "react";  import styled from "styled-components";  const Button = styled**.button**`  .button {  font: inherit;  padding: 0.5rem 1.5rem;  border: 1px solid #8b005d;  color: white;  background: #8b005d;  box-shadow: 0 0 4px rgba(0, 0, 0, 0.26);  cursor: pointer;  }  .button:focus {  outline: none;  }  .button:hover,  .button:active {  background: #ac0e77;  border-color: #ac0e77;  box-shadow: 0 0 8px rgba(0, 0, 0, 0.26);  }  `;  // const Button = props => {  // return (  // <button type={props.type} className="button" onClick={props.onClick}>  // {props.children}  // </button>  // );  // };  export default Button; |

You therefore get rid of any selectors like this

|  |
| --- |
| src/components/UI/Button/Button.js |
| import React from "react";  import styled from "styled-components";  const Button = styled.button`  **~~.button~~** {  font: inherit;  padding: 0.5rem 1.5rem;  border: 1px solid #8b005d;  color: white;  background: #8b005d;  box-shadow: 0 0 4px rgba(0, 0, 0, 0.26);  cursor: pointer;  }  .button:focus {  outline: none;  }  .button:hover,  .button:active {  background: #ac0e77;  border-color: #ac0e77;  box-shadow: 0 0 8px rgba(0, 0, 0, 0.26);  }  `;  // const Button = props => {  // return (  // <button type={props.type} className="button" onClick={props.onClick}>  // {props.children}  // </button>  // );  // };  export default Button; |

And, instead, just have your styles like this, and they will be added to the button element, which is returned by

|  |
| --- |
| src/components/UI/Button/Button.js |
| import React from "react";  import styled from "styled-components";  const Button = styled.button`  **font: inherit;**  **padding: 0.5rem 1.5rem;**  **border: 1px solid #8b005d;**  **color: white;**  **background: #8b005d;**  **box-shadow: 0 0 4px rgba(0, 0, 0, 0.26);**  **cursor: pointer;**  .button:focus {  outline: none;  }  .button:hover,  .button:active {  background: #ac0e77;  border-color: #ac0e77;  box-shadow: 0 0 8px rgba(0, 0, 0, 0.26);  }  `;  // const Button = props => {  // return (  // <button type={props.type} className="button" onClick={props.onClick}>  // {props.children}  // </button>  // );  // };  export default Button; |

this method call

|  |
| --- |
| src/components/UI/Button/Button.js |
| import React from "react";  import styled from "styled-components";  const Button = **styled.button**`  font: inherit;  padding: 0.5rem 1.5rem;  border: 1px solid #8b005d;  color: white;  background: #8b005d;  box-shadow: 0 0 4px rgba(0, 0, 0, 0.26);  cursor: pointer;  .button:focus {  outline: none;  }  .button:hover,  .button:active {  background: #ac0e77;  border-color: #ac0e77;  box-shadow: 0 0 8px rgba(0, 0, 0, 0.26);  }  `;  // const Button = props => {  // return (  // <button type={props.type} className="button" onClick={props.onClick}>  // {props.children}  // </button>  // );  // };  export default Button; |

which is then stored in this Button constant

|  |
| --- |
| src/components/UI/Button/Button.js |
| import React from "react";  import styled from "styled-components";  const **Button** = styled.button`  font: inherit;  padding: 0.5rem 1.5rem;  border: 1px solid #8b005d;  color: white;  background: #8b005d;  box-shadow: 0 0 4px rgba(0, 0, 0, 0.26);  cursor: pointer;  .button:focus {  outline: none;  }  .button:hover,  .button:active {  background: #ac0e77;  border-color: #ac0e77;  box-shadow: 0 0 8px rgba(0, 0, 0, 0.26);  }  `;  // const Button = props => {  // return (  // <button type={props.type} className="button" onClick={props.onClick}>  // {props.children}  // </button>  // );  // };  export default Button; |

For pseudo-selectors, you can use the ampersand symbol because it's supported by the 'styled-components' package. The ampersand tells the package that you want to have a special case, a special pseudo-selector in this case here, for the Button component, which is created here.

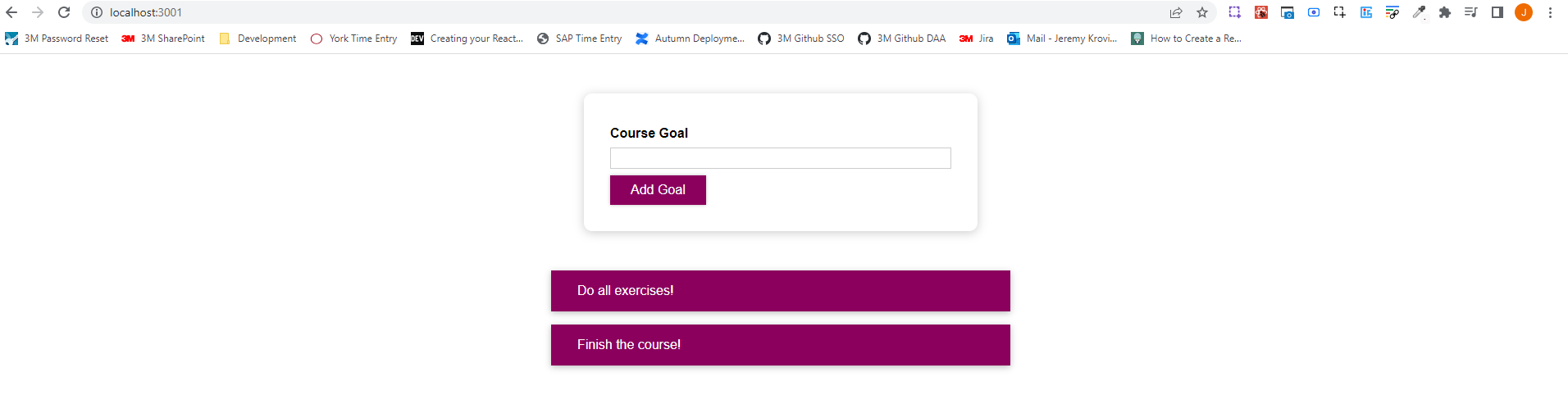
So, in the bolded part of the code snippet below, we're saying when this Button has focus, please apply this style. We do the same for hover and for active.

|  |
| --- |
| src/components/UI/Button/Button.js |
| import React from "react";  import styled from "styled-components";  const Button = styled.button`  font: inherit;  padding: 0.5rem 1.5rem;  border: 1px solid #8b005d;  color: white;  background: #8b005d;  box-shadow: 0 0 4px rgba(0, 0, 0, 0.26);  cursor: pointer;  **&:focus {**  **outline: none;**  **}**  &:hover,  &:active {  background: #ac0e77;  border-color: #ac0e77;  box-shadow: 0 0 8px rgba(0, 0, 0, 0.26);  }  `;  // const Button = props => {  // return (  // <button type={props.type} className="button" onClick={props.onClick}>  // {props.children}  // </button>  // );  // };  export default Button; |

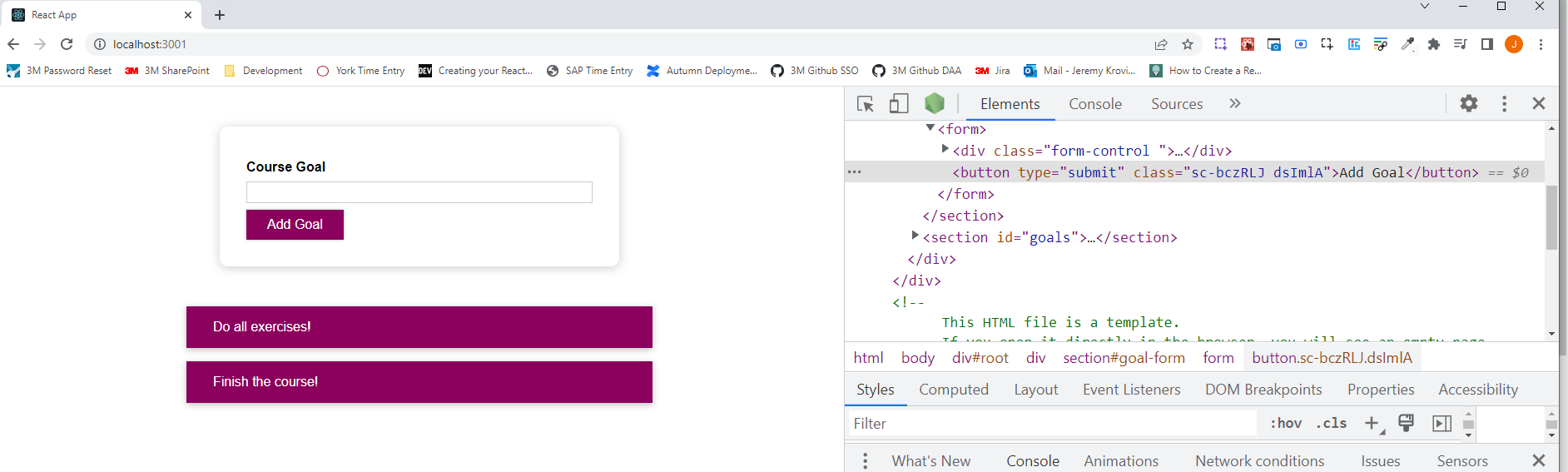
Now the button, which has all these styles by default applies all the props you might be passing to your own Button component, which we're exporting. Therefore, we can still add an onClick prop, we can still set the type. That will all be forwarded by the "styled-components" package to that core built-in button. We can now get rid of import React from "react" because we are no longer dealing with JSX in the "Button.js" file anymore.

|  |
| --- |
| src/components/UI/Button/Button.js |
| import styled from "styled-components";  const Button = styled.button`  font: inherit;  padding: 0.5rem 1.5rem;  border: 1px solid #8b005d;  color: white;  background: #8b005d;  box-shadow: 0 0 4px rgba(0, 0, 0, 0.26);  cursor: pointer;  &:focus {  outline: none;  }  &:hover,  &:active {  background: #ac0e77;  border-color: #ac0e77;  box-shadow: 0 0 8px rgba(0, 0, 0, 0.26);  }  `;  // const Button = props => {  // return (  // <button type={props.type} className="button" onClick={props.onClick}>  // {props.children}  // </button>  // );  // };  export default Button; |

If we now save this, we should see that our application looks and works exactly as it did before.



If we inspect the our styled button in the developer tools, we can tell that there is something different. Of course it's a regular button, but we see that there are two strangely named classes. These are class names that are dynamically generated by the 'styled-components' package because in the end the 'styled-components' package looks at the styles that we have set up and then it wraps the styles into generated class names where it guarantees that every class name is unique so it can't spill over to other parts of the app. And then it will add the classes as global CSS. But now since we have unique class names for every styled component, the style setup in "Button.js" will never be able to affect another component in the App because of these unique class names that were generated.



### 78. Styled Components & Dynamic Props

'styled-components' are really useful, but can we use them for our div here, for example?

|  |
| --- |
| src/components/CourseGoals/CourseInput/CourseInput.js |
| import React, { useState } from "react";  import Button from "../../UI/Button/Button";  import "./CourseInput.css";  const CourseInput = (props) => {  const [enteredValue, setEnteredValue] = useState("");  const [isValid, setIsValid] = useState(true);  const goalInputChangeHandler = (event) => {  if (event.target.value.trim().length > 0) {  setIsValid(true);  }  setEnteredValue(event.target.value);  };  const formSubmitHandler = (event) => {  event.preventDefault();  if (enteredValue.trim().length === 0) {  setIsValid(false);  return;  }  props.onAddGoal(enteredValue);  };  return (  <form onSubmit={formSubmitHandler}>  **<div className={`form-control ${!isValid ? "invalid" : ""}`}>**  **<label>Course Goal</label>**  **<input type="text" onChange={goalInputChangeHandler} />**  **</div>**  <Button type="submit">Add Goal</Button>  </form>  );  };  export default CourseInput; |

We absolutely can use "styled-components" for our div here. If we want a styled-component that we only use in "CourseInput.js", we can also create it in that file. We can also create a brand new file as we did for the Button. Even though, we're only using the Button here, we could still argue that a Button is a component, as you might be using it in a lot of other places in the app as well.

|  |
| --- |
| src/components/CourseGoals/CourseInput/CourseInput.js |
| import React, { useState } from "react";  import Button from "../../UI/Button/Button";  import "./CourseInput.css";  const CourseInput = (props) => {  const [enteredValue, setEnteredValue] = useState("");  const [isValid, setIsValid] = useState(true);  const goalInputChangeHandler = (event) => {  if (event.target.value.trim().length > 0) {  setIsValid(true);  }  setEnteredValue(event.target.value);  };  const formSubmitHandler = (event) => {  event.preventDefault();  if (enteredValue.trim().length === 0) {  setIsValid(false);  return;  }  props.onAddGoal(enteredValue);  };  return (  <form onSubmit={formSubmitHandler}>  <div className={`form-control ${!isValid ? "invalid" : ""}`}>  <label>Course Goal</label>  <input type="text" onChange={goalInputChangeHandler} />  </div>  **<Button type="submit">Add Goal</Button>**  </form>  );  };  export default CourseInput; |

For the div, that also might be the case. You might have other form inputs in other parts of the App as well.

|  |
| --- |
| src/components/CourseGoals/CourseInput/CourseInput.js |
| import React, { useState } from "react";  import Button from "../../UI/Button/Button";  import "./CourseInput.css";  const CourseInput = (props) => {  const [enteredValue, setEnteredValue] = useState("");  const [isValid, setIsValid] = useState(true);  const goalInputChangeHandler = (event) => {  if (event.target.value.trim().length > 0) {  setIsValid(true);  }  setEnteredValue(event.target.value);  };  const formSubmitHandler = (event) => {  event.preventDefault();  if (enteredValue.trim().length === 0) {  setIsValid(false);  return;  }  props.onAddGoal(enteredValue);  };  return (  <form onSubmit={formSubmitHandler}>  **<div className={`form-control ${!isValid ? "invalid" : ""}`}>**  **<label>Course Goal</label>**  **<input type="text" onChange={goalInputChangeHandler} />**  **</div>**  <Button type="submit">Add Goal</Button>  </form>  );  };  export default CourseInput; |

To show you that you can have multiple components per file and that this might make sense if a component is really only getting used in that file, we will create a new component in the file "CourseInput.js". Thus far, we have only had one component per file, and that generally is a good rule to keep, but if you have a component that is only really used by the other component in that file, having both components in the same file can make sense, too.

We import styled from 'styled-components';

|  |
| --- |
| src/components/CourseGoals/CourseInput/CourseInput.js |
| import React, { useState } from "react";  **import styled from 'styled-components';**  import Button from "../../UI/Button/Button";  import "./CourseInput.css";  const CourseInput = (props) => {  const [enteredValue, setEnteredValue] = useState("");  const [isValid, setIsValid] = useState(true);  const goalInputChangeHandler = (event) => {  if (event.target.value.trim().length > 0) {  setIsValid(true);  }  setEnteredValue(event.target.value);  };  const formSubmitHandler = (event) => {  event.preventDefault();  if (enteredValue.trim().length === 0) {  setIsValid(false);  return;  }  props.onAddGoal(enteredValue);  };  return (  <form onSubmit={formSubmitHandler}>  <div className={`form-control ${!isValid ? "invalid" : ""}`}>  <label>Course Goal</label>  <input type="text" onChange={goalInputChangeHandler} />  </div>  <Button type="submit">Add Goal</Button>  </form>  );  };  export default CourseInput; |

I create a component called FormControl, for example, the name is ultimately up to me but since we are creating it as a component in JSX, it must start with a capital character. We then use styled.div because we want to replace

|  |
| --- |
| src/components/CourseGoals/CourseInput/CourseInput.js |
| import React, { useState } from "react";  import styled from 'styled-components';  import Button from "../../UI/Button/Button";  import "./CourseInput.css";  **const FormControl = styled.div``;**  const CourseInput = (props) => {  const [enteredValue, setEnteredValue] = useState("");  const [isValid, setIsValid] = useState(true);  const goalInputChangeHandler = (event) => {  if (event.target.value.trim().length > 0) {  setIsValid(true);  }  setEnteredValue(event.target.value);  };  const formSubmitHandler = (event) => {  event.preventDefault();  if (enteredValue.trim().length === 0) {  setIsValid(false);  return;  }  props.onAddGoal(enteredValue);  };  return (  <form onSubmit={formSubmitHandler}>  <div className={`form-control ${!isValid ? "invalid" : ""}`}>  <label>Course Goal</label>  <input type="text" onChange={goalInputChangeHandler} />  </div>  <Button type="submit">Add Goal</Button>  </form>  );  };  export default CourseInput; |

this div later.

|  |
| --- |
| src/components/CourseGoals/CourseInput/CourseInput.js |
| import React, { useState } from "react";  import styled from 'styled-components';  import Button from "../../UI/Button/Button";  import "./CourseInput.css";  const FormControl = styled.div``;  const CourseInput = (props) => {  const [enteredValue, setEnteredValue] = useState("");  const [isValid, setIsValid] = useState(true);  const goalInputChangeHandler = (event) => {  if (event.target.value.trim().length > 0) {  setIsValid(true);  }  setEnteredValue(event.target.value);  };  const formSubmitHandler = (event) => {  event.preventDefault();  if (enteredValue.trim().length === 0) {  setIsValid(false);  return;  }  props.onAddGoal(enteredValue);  };  return (  <form onSubmit={formSubmitHandler}>  **<div className={`form-control ${!isValid ? "invalid" : ""}`}>**  **<label>Course Goal</label>**  **<input type="text" onChange={goalInputChangeHandler} />**  **</div>**  <Button type="submit">Add Goal</Button>  </form>  );  };  export default CourseInput; |

Within the backticks, we want to add the styles for the form-control class. So we copy them from "CourseInput.css" and paste them between the backticks and get rid of the selectors. If you want to target a nested element, you can add the ampersand, which tells 'styled-components' that we are now talking about a label inside of the div.

|  |
| --- |
| src/components/CourseGoals/CourseInput/CourseInput.js |
| import React, { useState } from "react";  import styled from "styled-components";  import Button from "../../UI/Button/Button";  import "./CourseInput.css";  const FormControl = styled.div`  **margin: 0.5rem 0;**  **& label {**  **font-weight: bold;**  **display: block;**  **margin-bottom: 0.5rem;**  **}**  **& input {**  **display: block;**  **width: 100%;**  **border: 1px solid #ccc;**  **font: inherit;**  **line-height: 1.5rem;**  **padding: 0 0.25rem;**  **}**  **& input:focus {**  **outline: none;**  **background: #fad0ec;**  **border-color: #8b005d;**  **}**  **&.invalid input {**  **border-color: red;**  **background: #ffd7d7;**  **}**  **&.invalid label {**  **color: red;**  **}**  `;  const CourseInput = (props) => {  const [enteredValue, setEnteredValue] = useState("");  const [isValid, setIsValid] = useState(true);  const goalInputChangeHandler = (event) => {  if (event.target.value.trim().length > 0) {  setIsValid(true);  }  setEnteredValue(event.target.value);  };  const formSubmitHandler = (event) => {  event.preventDefault();  if (enteredValue.trim().length === 0) {  setIsValid(false);  return;  }  props.onAddGoal(enteredValue);  };  return (  <form onSubmit={formSubmitHandler}>  <div className={`form-control ${!isValid ? "invalid" : ""}`}>  <label>Course Goal</label>  <input type="text" onChange={goalInputChangeHandler} />  </div>  <Button type="submit">Add Goal</Button>  </form>  );  };  export default CourseInput; |

We can now go and replace the div tags below with our FormControl.

|  |
| --- |
| src/components/CourseGoals/CourseInput/CourseInput.js |
| import React, { useState } from "react";  import styled from "styled-components";  import Button from "../../UI/Button/Button";  import "./CourseInput.css";  const FormControl = styled.div`  margin: 0.5rem 0;  & label {  font-weight: bold;  display: block;  margin-bottom: 0.5rem;  }  & input {  display: block;  width: 100%;  border: 1px solid #ccc;  font: inherit;  line-height: 1.5rem;  padding: 0 0.25rem;  }  & input:focus {  outline: none;  background: #fad0ec;  border-color: #8b005d;  }  &.invalid input {  border-color: red;  background: #ffd7d7;  }  &.invalid label {  color: red;  }  `;  const CourseInput = (props) => {  const [enteredValue, setEnteredValue] = useState("");  const [isValid, setIsValid] = useState(true);  const goalInputChangeHandler = (event) => {  if (event.target.value.trim().length > 0) {  setIsValid(true);  }  setEnteredValue(event.target.value);  };  const formSubmitHandler = (event) => {  event.preventDefault();  if (enteredValue.trim().length === 0) {  setIsValid(false);  return;  }  props.onAddGoal(enteredValue);  };  return (  <form onSubmit={formSubmitHandler}>  **<FormControl>**  <label>Course Goal</label>  <input type="text" onChange={goalInputChangeHandler} />  **</FormControl>**  <Button type="submit">Add Goal</Button>  </form>  );  };  export default CourseInput; |

If we save it, it generally works, but if we try to submit it with an empty input, we don't get the error. Of course, it cannot work because we are no longer setting the invalid class. The good thing is that the 'styled-components' forward all props you set on them to the underlying components, so to the underlying div, so we can still add a className to our FormControl component tag, and the value that we want to pass in is either invalid or an empty string. We can simply check if not isValid, and if that's the case, we add the string 'invalid', and with the && syntax we otherwise add nothing, which is perfectly fine because we need no other class if it is valid. But if it is not valid, the "invalid" class is being added. Now, we have the same behavior as we did before.

|  |
| --- |
| src/components/CourseGoals/CourseInput/CourseInput.js |
| import React, { useState } from "react";  import styled from "styled-components";  import Button from "../../UI/Button/Button";  import "./CourseInput.css";  const FormControl = styled.div`  margin: 0.5rem 0;  & label {  font-weight: bold;  display: block;  margin-bottom: 0.5rem;  }  & input {  display: block;  width: 100%;  border: 1px solid #ccc;  font: inherit;  line-height: 1.5rem;  padding: 0 0.25rem;  }  & input:focus {  outline: none;  background: #fad0ec;  border-color: #8b005d;  }  &.invalid input {  border-color: red;  background: #ffd7d7;  }  &.invalid label {  color: red;  }  `;  const CourseInput = (props) => {  const [enteredValue, setEnteredValue] = useState("");  const [isValid, setIsValid] = useState(true);  const goalInputChangeHandler = (event) => {  if (event.target.value.trim().length > 0) {  setIsValid(true);  }  setEnteredValue(event.target.value);  };  const formSubmitHandler = (event) => {  event.preventDefault();  if (enteredValue.trim().length === 0) {  setIsValid(false);  return;  }  props.onAddGoal(enteredValue);  };  return (  <form onSubmit={formSubmitHandler}>  <FormControl className={**!isValid && 'invalid'**}>  <label>Course Goal</label>  <input type="text" onChange={goalInputChangeHandler} />  </FormControl>  <Button type="submit">Add Goal</Button>  </form>  );  };  export default CourseInput; |

That is only one way of doing this. There is also another way, where you can utilize another feature provided by 'styled-components'. You can also add props to your styled-components and utilize the props inside of the backticks, so inside of your styles to easily change styles dynamically. How does this work? Let's say when we use FormControl, we want to add an invalid prop. The value that we feed in should either be true or false. Therefore, we can do the following shown in the below code snippet in bold.

|  |
| --- |
| src/components/CourseGoals/CourseInput/CourseInput.js |
| import React, { useState } from "react";  import styled from "styled-components";  import Button from "../../UI/Button/Button";  import "./CourseInput.css";  const FormControl = styled.div`  margin: 0.5rem 0;  & label {  font-weight: bold;  display: block;  margin-bottom: 0.5rem;  }  & input {  display: block;  width: 100%;  border: 1px solid #ccc;  font: inherit;  line-height: 1.5rem;  padding: 0 0.25rem;  }  & input:focus {  outline: none;  background: #fad0ec;  border-color: #8b005d;  }  &.invalid input {  border-color: red;  background: #ffd7d7;  }  &.invalid label {  color: red;  }  `;  const CourseInput = (props) => {  const [enteredValue, setEnteredValue] = useState("");  const [isValid, setIsValid] = useState(true);  const goalInputChangeHandler = (event) => {  if (event.target.value.trim().length > 0) {  setIsValid(true);  }  setEnteredValue(event.target.value);  };  const formSubmitHandler = (event) => {  event.preventDefault();  if (enteredValue.trim().length === 0) {  setIsValid(false);  return;  }  props.onAddGoal(enteredValue);  };  return (  <form onSubmit={formSubmitHandler}>  <FormControl **invalid={!isValid}**>  <label>Course Goal</label>  <input type="text" onChange={goalInputChangeHandler} />  </FormControl>  <Button type="submit">Add Goal</Button>  </form>  );  };  export default CourseInput; |

If invalid is false we feed in true because invalid is true if isInvalid is not true, so we set invalid to true if what the user entered was invalid. And this invalid prop can now be used between the backticks. How do we use it? Well, we wanted to change the border color of the input, for example. So we go to the place where we set up the border color. And now here, since you're between backticks, you can use this ${} syntax you already know, pass in an arrow function, which receives props as a parameter ${props => } and then should return the text that should be returned in this exact position. This function ${props => } will be called by the 'styled-components' package, and for props it will feed in all the props your component gets, so, in this case, the invalid prop. We can now check if invalid is true. ${props => ( props.invalid ? ) }, and if invalid is true, we return red text ${props => ( props.invalid ? 'red' : )}; otherwise, we return '#ccc' ${props => ( props.invalid ? 'red' : '#ccc')}; So with that, we're having some dynamic way of changing parts of the styles based on the props that are passed to our 'styled-component'.

|  |
| --- |
| src/components/CourseGoals/CourseInput/CourseInput.js |
| import React, { useState } from "react";  import styled from "styled-components";  import Button from "../../UI/Button/Button";  import "./CourseInput.css";  const FormControl = styled.div`  margin: 0.5rem 0;  & label {  font-weight: bold;  display: block;  margin-bottom: 0.5rem;  }  & input {  display: block;  width: 100%;  border: 1px solid **${props => (props.invalid ? 'red' : 'black')}**;  font: inherit;  line-height: 1.5rem;  padding: 0 0.25rem;  }  & input:focus {  outline: none;  background: #fad0ec;  border-color: #8b005d;  }  &.invalid input {  border-color: red;  background: #ffd7d7;  }  &.invalid label {  color: red;  }  `;  const CourseInput = (props) => {  const [enteredValue, setEnteredValue] = useState("");  const [isValid, setIsValid] = useState(true);  const goalInputChangeHandler = (event) => {  if (event.target.value.trim().length > 0) {  setIsValid(true);  }  setEnteredValue(event.target.value);  };  const formSubmitHandler = (event) => {  event.preventDefault();  if (enteredValue.trim().length === 0) {  setIsValid(false);  return;  }  props.onAddGoal(enteredValue);  };  return (  <form onSubmit={formSubmitHandler}>  <FormControl invalid={!isValid}>  <label>Course Goal</label>  <input type="text" onChange={goalInputChangeHandler} />  </FormControl>  <Button type="submit">Add Goal</Button>  </form>  );  };  export default CourseInput; |

We also want to do this for background, so we set background: to a dynamic value ${} and just as before, I check if props is invalid: ${props => (props.invalid ? )} And if it is invalid, I set this background to this light redish color: #ffd7d7 ${props => (props.invalid ? '#ffd7d7' : )}; otherwise, we'll set the background to 'transparent' ${props => (props.invalid ? '#ffd7d7' : 'transparent')}. With that we can remove the selector: &.invalid input

|  |
| --- |
| src/components/CourseGoals/CourseInput/CourseInput.js |
| import React, { useState } from "react";  import styled from "styled-components";  import Button from "../../UI/Button/Button";  import "./CourseInput.css";  const FormControl = styled.div`  margin: 0.5rem 0;  & label {  font-weight: bold;  display: block;  margin-bottom: 0.5rem;  }  & input {  display: block;  width: 100%;  border: 1px solid ${(props) => (props.invalid ? "red" : "black")};  **background: ${(props) => (props.invalid ? "#ffd7d7" : "transparent")};**  font: inherit;  line-height: 1.5rem;  padding: 0 0.25rem;  }  & input:focus {  outline: none;  background: #fad0ec;  border-color: #8b005d;  }  ~~&.invalid input {~~  ~~border-color: red;~~  ~~background: #ffd7d7;~~  ~~}~~  &.invalid label {  color: red;  }  `;  const CourseInput = (props) => {  const [enteredValue, setEnteredValue] = useState("");  const [isValid, setIsValid] = useState(true);  const goalInputChangeHandler = (event) => {  if (event.target.value.trim().length > 0) {  setIsValid(true);  }  setEnteredValue(event.target.value);  };  const formSubmitHandler = (event) => {  event.preventDefault();  if (enteredValue.trim().length === 0) {  setIsValid(false);  return;  }  props.onAddGoal(enteredValue);  };  return (  <form onSubmit={formSubmitHandler}>  <FormControl invalid={!isValid}>  <label>Course Goal</label>  <input type="text" onChange={goalInputChangeHandler} />  </FormControl>  <Button type="submit">Add Goal</Button>  </form>  );  };  export default CourseInput; |

&.invalid input removed

|  |
| --- |
| src/components/CourseGoals/CourseInput/CourseInput.js |
| import React, { useState } from "react";  import styled from "styled-components";  import Button from "../../UI/Button/Button";  import "./CourseInput.css";  const FormControl = styled.div`  margin: 0.5rem 0;  & label {  font-weight: bold;  display: block;  margin-bottom: 0.5rem;  }  & input {  display: block;  width: 100%;  border: 1px solid ${(props) => (props.invalid ? "red" : "black")};  background: ${(props) => (props.invalid ? "#ffd7d7" : "transparent")};  font: inherit;  line-height: 1.5rem;  padding: 0 0.25rem;  }  & input:focus {  outline: none;  background: #fad0ec;  border-color: #8b005d;  }  &.invalid label {  color: red;  }  `;  const CourseInput = (props) => {  const [enteredValue, setEnteredValue] = useState("");  const [isValid, setIsValid] = useState(true);  const goalInputChangeHandler = (event) => {  if (event.target.value.trim().length > 0) {  setIsValid(true);  }  setEnteredValue(event.target.value);  };  const formSubmitHandler = (event) => {  event.preventDefault();  if (enteredValue.trim().length === 0) {  setIsValid(false);  return;  }  props.onAddGoal(enteredValue);  };  return (  <form onSubmit={formSubmitHandler}>  <FormControl invalid={!isValid}>  <label>Course Goal</label>  <input type="text" onChange={goalInputChangeHandler} />  </FormControl>  <Button type="submit">Add Goal</Button>  </form>  );  };  export default CourseInput; |

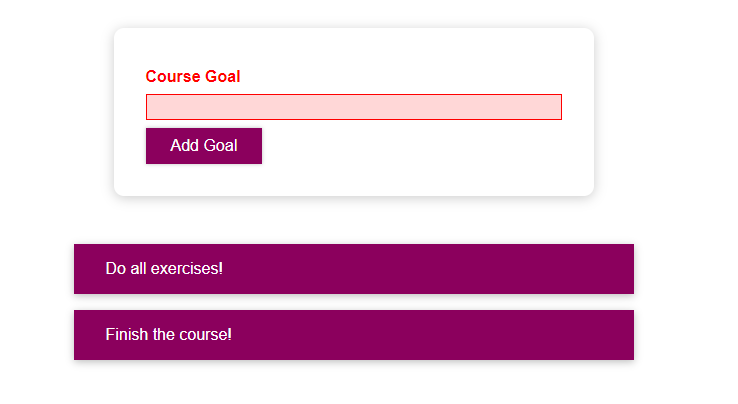
We can also remove the &.invalid label style, which was based on the existence of the "invalid" class, which we no longer add.

|  |
| --- |
| src/components/CourseGoals/CourseInput/CourseInput.js |
| import React, { useState } from "react";  import styled from "styled-components";  import Button from "../../UI/Button/Button";  import "./CourseInput.css";  const FormControl = styled.div`  margin: 0.5rem 0;  & label {  font-weight: bold;  display: block;  margin-bottom: 0.5rem;  }  & input {  display: block;  width: 100%;  border: 1px solid ${(props) => (props.invalid ? "red" : "black")};  background: ${(props) => (props.invalid ? "#ffd7d7" : "transparent")};  font: inherit;  line-height: 1.5rem;  padding: 0 0.25rem;  }  & input:focus {  outline: none;  background: #fad0ec;  border-color: #8b005d;  }  `;  const CourseInput = (props) => {  const [enteredValue, setEnteredValue] = useState("");  const [isValid, setIsValid] = useState(true);  const goalInputChangeHandler = (event) => {  if (event.target.value.trim().length > 0) {  setIsValid(true);  }  setEnteredValue(event.target.value);  };  const formSubmitHandler = (event) => {  event.preventDefault();  if (enteredValue.trim().length === 0) {  setIsValid(false);  return;  }  props.onAddGoal(enteredValue);  };  return (  <form onSubmit={formSubmitHandler}>  <FormControl invalid={!isValid}>  <label>Course Goal</label>  <input type="text" onChange={goalInputChangeHandler} />  </FormControl>  <Button type="submit">Add Goal</Button>  </form>  );  };  export default CourseInput; |

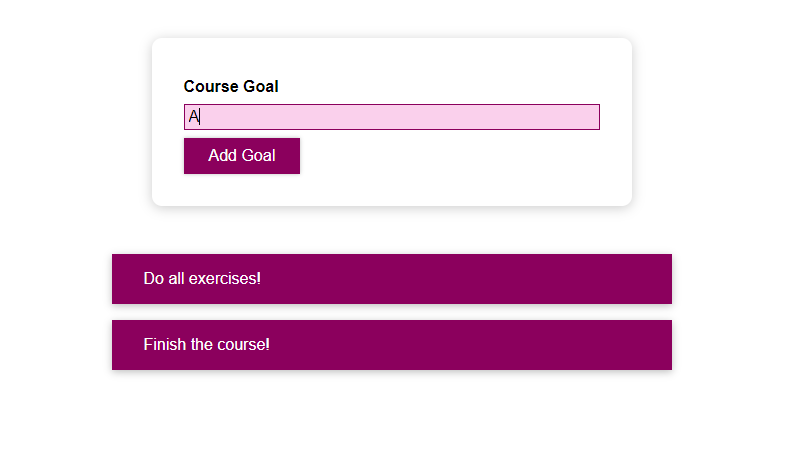
And we can instead go up to the & label styling and add a color definition. And just as before, the value we set for color is dynamic. We provide this function ${props => }; which will be called by the 'styled-components' package, and we check if the invalid props is true. ${props => (props.invalid ? )}; If the invalid prop is true. If it is true, we set the color 'red': ${props => (props.invalid ? 'red' : )}; otherwise, we set a color of 'black': ${props => (props.invalid ? 'red' : 'black')};

|  |
| --- |
| src/components/CourseGoals/CourseInput/CourseInput.js |
| import React, { useState } from "react";  import styled from "styled-components";  import Button from "../../UI/Button/Button";  import "./CourseInput.css";  const FormControl = styled.div`  margin: 0.5rem 0;  & label {  font-weight: bold;  display: block;  margin-bottom: 0.5rem;  **color: ${(props) => (props.invalid ? "red" : "black")};**  }  & input {  display: block;  width: 100%;  border: 1px solid ${(props) => (props.invalid ? "red" : "black")};  background: ${(props) => (props.invalid ? "#ffd7d7" : "transparent")};  font: inherit;  line-height: 1.5rem;  padding: 0 0.25rem;  }  & input:focus {  outline: none;  background: #fad0ec;  border-color: #8b005d;  }  `;  const CourseInput = (props) => {  const [enteredValue, setEnteredValue] = useState("");  const [isValid, setIsValid] = useState(true);  const goalInputChangeHandler = (event) => {  if (event.target.value.trim().length > 0) {  setIsValid(true);  }  setEnteredValue(event.target.value);  };  const formSubmitHandler = (event) => {  event.preventDefault();  if (enteredValue.trim().length === 0) {  setIsValid(false);  return;  }  props.onAddGoal(enteredValue);  };  return (  <form onSubmit={formSubmitHandler}>  <FormControl invalid={!isValid}>  <label>Course Goal</label>  <input type="text" onChange={goalInputChangeHandler} />  </FormControl>  <Button type="submit">Add Goal</Button>  </form>  );  };  export default CourseInput; |

When we save and reload the page. If we fail to enter in any input, we have the same behavior as we had before when we failed to enter any input.



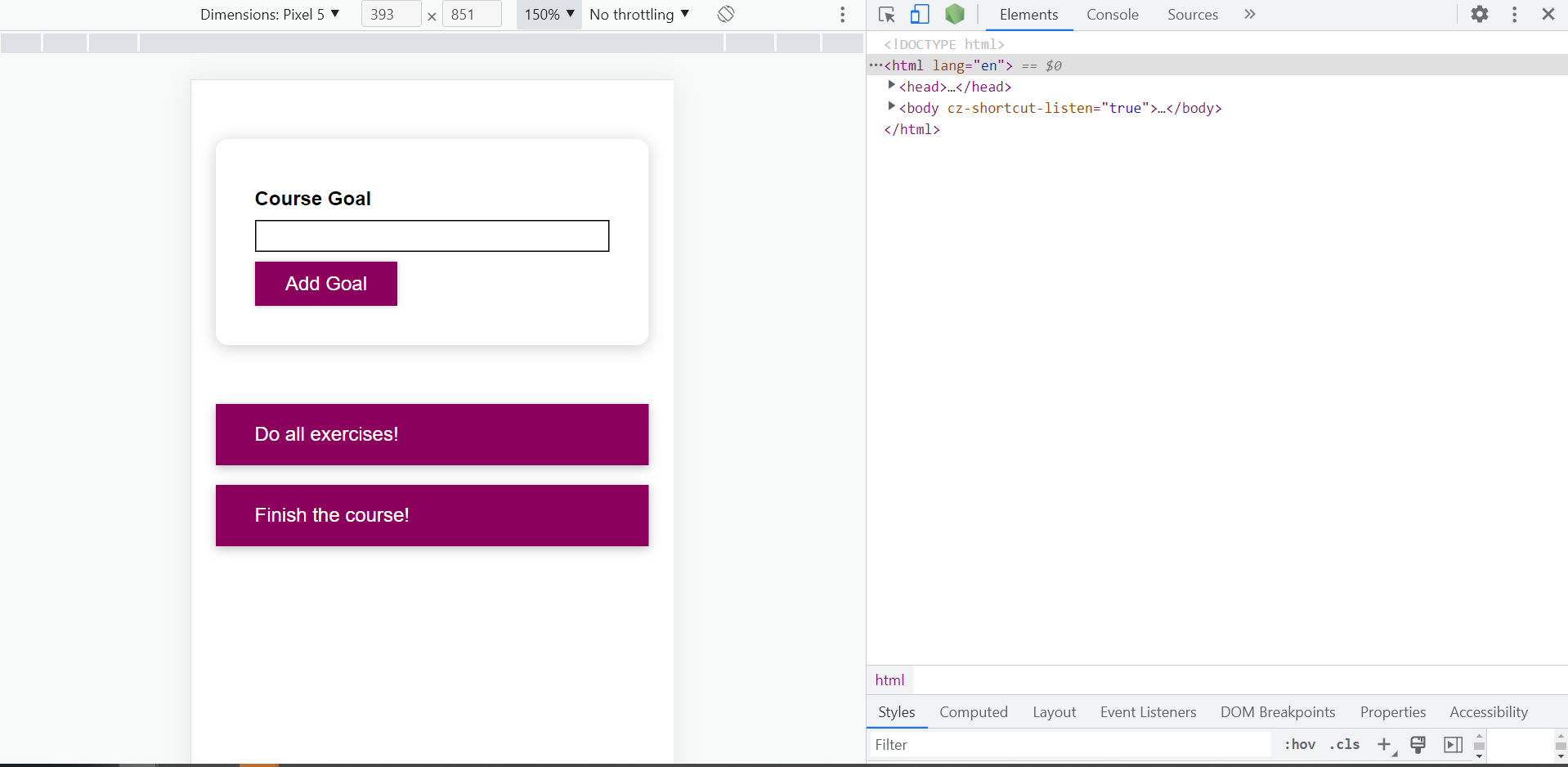
And when we start typing, we see the style change back.



We dynamically changed parts of the styles based on the props that are set on your styled component.

### 79. Styled Components & Media Queries

Media queries of course are often important to get the right look. At the moment, if we preview our project on a mobile device, on the Pixel 5 let's say. This looks good from a pure styling perspective. Nothing is going beyond the edges, so it looks okay. But maybe we want to make sure that the "Add Goal" button actually spans the entire width, if we're on a small device, and only takes the space it needs if we're not on a mobile device. That would be a scenario where we might want to add a media query.



How would we add a media query when using styled components? You add @media and define your criteria: @media (min-width: 768 px){}, and now we just put the styles into that media query that should affect this element, when that condition, min-width of 768px, is met. That's all. You need no selector in there, nothing else, just the styles that should be applied to the Button component if this condition for this media query here is met.

|  |
| --- |
| src/components/UI/Button/Button.js |
| import styled from "styled-components";  const Button = styled.button`  font: inherit;  padding: 0.5rem 1.5rem;  border: 1px solid #8b005d;  color: white;  background: #8b005d;  box-shadow: 0 0 4px rgba(0, 0, 0, 0.26);  cursor: pointer;  **@media (min-width: 768px) {**  **}**  &:focus {  outline: none;  }  &:hover,  &:active {  background: #ac0e77;  border-color: #ac0e77;  box-shadow: 0 0 8px rgba(0, 0, 0, 0.26);  }  `;  // const Button = props => {  // return (  // <button type={props.type} className="button" onClick={props.onClick}>  // {props.children}  // </button>  // );  // };  export default Button; |

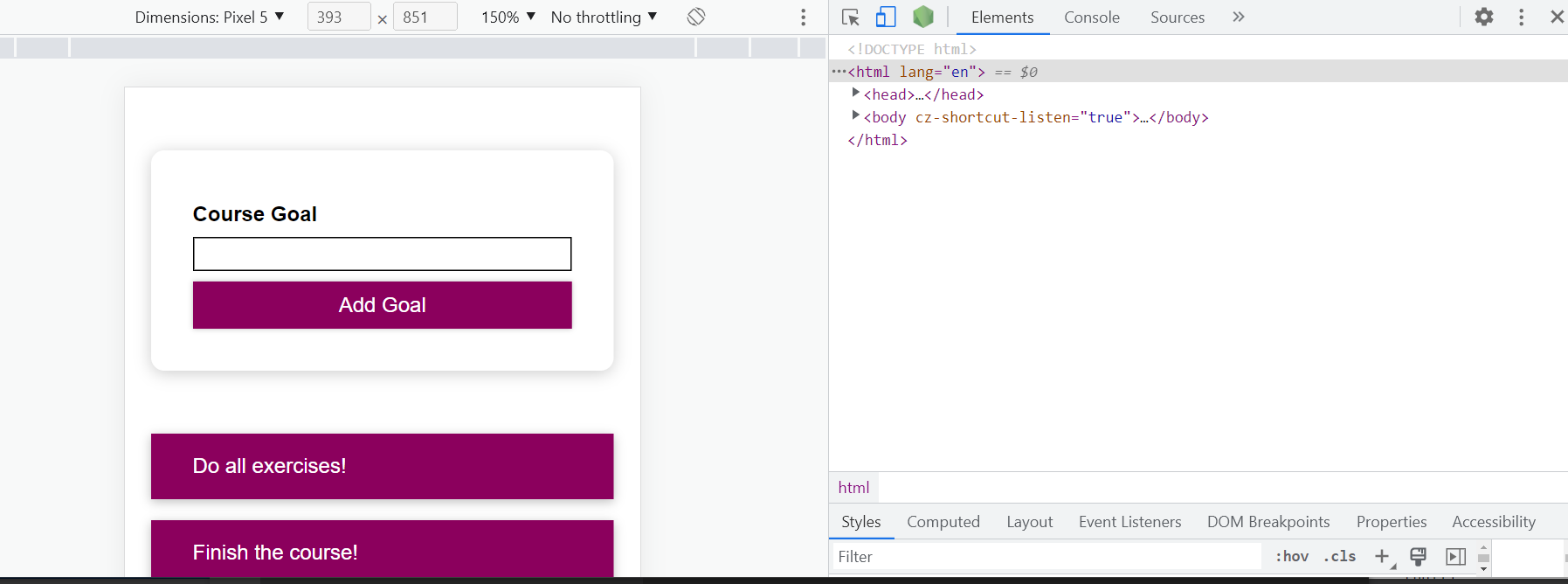
So, here that is actually the media query for big devices, non-mobile devices, and there I want to set the width to auto.

|  |
| --- |
| src/components/UI/Button/Button.js |
| import styled from "styled-components";  const Button = styled.button`  font: inherit;  padding: 0.5rem 1.5rem;  border: 1px solid #8b005d;  color: white;  background: #8b005d;  box-shadow: 0 0 4px rgba(0, 0, 0, 0.26);  cursor: pointer;  @media (min-width: 768px) {  **width: auto;**  }  &:focus {  outline: none;  }  &:hover,  &:active {  background: #ac0e77;  border-color: #ac0e77;  box-shadow: 0 0 8px rgba(0, 0, 0, 0.26);  }  `;  // const Button = props => {  // return (  // <button type={props.type} className="button" onClick={props.onClick}>  // {props.children}  // </button>  // );  // };  export default Button; |

The default width, so if the media query is not met, can be set to 100%.

|  |
| --- |
| src/components/UI/Button/Button.js |
| import styled from "styled-components";  const Button = styled.button`  **width: 100%;**  font: inherit;  padding: 0.5rem 1.5rem;  border: 1px solid #8b005d;  color: white;  background: #8b005d;  box-shadow: 0 0 4px rgba(0, 0, 0, 0.26);  cursor: pointer;  @media (min-width: 768px) {  width: auto;  }  &:focus {  outline: none;  }  &:hover,  &:active {  background: #ac0e77;  border-color: #ac0e77;  box-shadow: 0 0 8px rgba(0, 0, 0, 0.26);  }  `;  // const Button = props => {  // return (  // <button type={props.type} className="button" onClick={props.onClick}>  // {props.children}  // </button>  // );  // };  export default Button; |

As a result, if we save this, we have this big button "Add Goal" for a mobile device,



but if we go to a bigger screen, the "Add Goal" button only takes as much space as it needs.

